REPOSITORIOS DSPACE CON MÚLTIPLES CONTEXTOS OAI-PMH

De Giusti Marisa Raquel[1], Adorno Facundo Gabriel[2], Lira Ariel Jorge[3]

1. Ingeniera en Telecomunicaciones y Profesora de Letras. Directora del Servicio de Difusión de la Creación Intelectual (SEDICI) y del Proyecto de Enlace de Bibliotecas (PREBI), Universidad Nacional de La Plata (UNLP) e Investigador de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CICBA).
2. Estudiante avanzado en la Facultad de Informática, UNLP. SEDICI, UNLP.
3. Licenciado en Informática. SEDICI, PREBI, UNLP.

Introducción

En la actualidad, casi la mitad de los repositorios digitales del mundo están soportados por el software DSpace[13] y lo mismo sucede en igual o mayor medida con los nuevos repositorios que se siguen creando día a día. Esto puede atribuirse entre otros motivos a que es un desarrollo de código abierto, a la funcionalidad que provee, a la gran cantidad de documentación y experiencias en línea sobre su uso e instalación y en particular a su gran comunidad de usuarios y desarrolladores que constantemente lo actualiza y expande.
El proyecto DSpace posee una arquitectura[12] compleja que se divide en 3 grandes capas o áreas: aplicación, lógica de negocios y almacenamiento. Ver Figura 1.

FIGURA 1 - Arquitectura DSpace

La capa de Aplicación incluye todas las herramientas que permiten al exterior (usuarios u otros sistemas) hacer uso del repositorio; por ejemplo XMLUI, JSPUI, módulo OAI, SWORD Server, entre otros. La capa intermedia mantiene la lógica transversal a todas las aplicaciones y que rige el funcionamiento interno del repositorio. Finalmente la capa de almacenamiento se encarga de todas las tareas específicas de guardado y recuperación desde almacenamiento secundario, es decir, bases de datos y sistema de archivos.
Una instalación DSpace ofrece varios puntos de configuración y extensión mediante diversos archivos, que definen parte del modelo de datos del repositorio, los flujos de trabajo, las características de interfaz y muchísimos puntos más. En ciertos casos, la forma y sintaxis de estos archivos son simples y permiten entender su lógica sin mayor complejidad. Sin embargo, muchos de estos archivos requieren no solo comprender el funcionamiento de la aplicación sino también entender su implementación y otros conceptos vinculados a las herramientas complementarias que utiliza DSpace para funcionar. Algunos ejemplos de éstos casos son la customización del proceso de ingesta o *submission*, del formulario de carga, de la configuración avanzada del módulo de búsqueda, entre otros.
Este trabajo se centra en aquellas tecnologías que brindan un servicio de interoperabilidad sobre el protocolo OAI-PMH[8] mediante el módulo OAI2.0 provisto por DSpace[1]. Dado que la interoperabilidad determina la capacidad de intercambiar información entre dos o más sistemas, este módulo se localiza en la capa de aplicación de DSpace, en contacto con

el mundo exterior. De acuerdo a estudios realizados, OAI-PMH es el protocolo más utilizado para implementar interoperabilidad de metadatos en el ámbito de repositorios digitales [15]. OAI-PMH provee un marco de interoperabilidad independiente de la aplicación subyacente y basado en la cosecha de metadatos. El protocolo define 2 roles claros, uno de *data provider*, correspondiente a la fuente que expone metadatos usando el protocolo, y otro denominado *service provider*, que se aplica a los sistemas externos que toman estos metadatos de uno o más *data providers*, los procesan y generan un servicio de valor agregado a partir de los mismos. En ocasiones, un repositorio que posee una red de repositorios colaboradores puede cumplir ambos roles. El módulo DSpace que se responsabiliza de la interoperabilidad mediante OAI-PMH es denominado OAI 2.0.
La experimentación que antecedió la realización de este trabajo se realizó en el repositorio institucional de la UNLP: el Servicio de Difusión de la Creación Intelectual (SEDICI)[3]. Actualmente, el repositorio cuenta con gran cantidad de publicaciones, más de 35 mil obras, que se exponen desde el módulo OAI 2.0 bajo 3 contextos de cosecha, correspondientes a las directrices de interoperabilidad OpenAIRE[4][5], DRIVER[6] y SNRD[7]. En las secciones siguientes se incluye un breve análisis sobre las decisiones tomadas y configuraciones usadas para la adecuación de este módulo a las 3 directrices de interoperabilidad.

Directrices

Las directrices analizadas están muy vinculadas dado que todas comparten a DRIVER2 como raíz y por tanto presentan similitudes en sus requerimientos y estructura. Ver Figura 2.

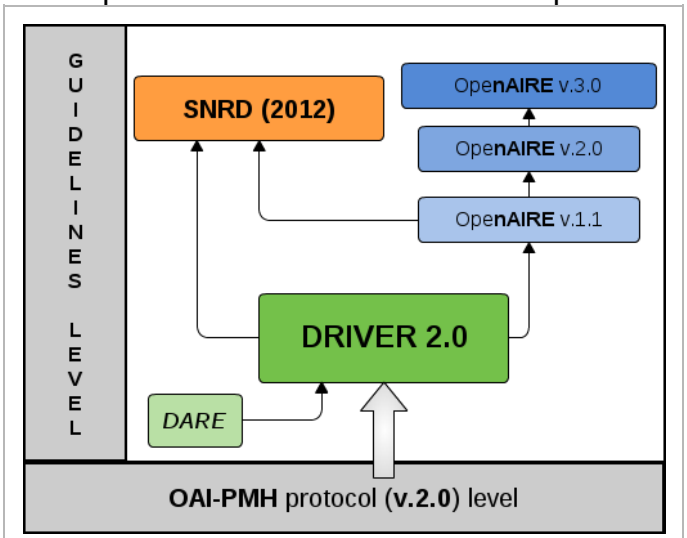

FIGURA 2 - Relaciones entre directrices

De este modo, varios de los aspectos de algunas de estas directrices están basados en los aspectos de otras; por ejemplo, las tipologías documentales usadas en DRIVER son también utilizadas por OpenAIRE. Según se indica en sus especificaciones, las directrices SNRD toman elementos de las directrices DRIVER y OpenAIRE 1.1; y las directrices OpenAIRE 1.1 se basan en DRIVER —las actuales directrices OpenAIRE 3.0 también—. Consecuentemente, SNRD y OpenAIRE 1.1 presentan características muy similares y se asemejan mucho a DRIVER.
Un objetivo común en estas directrices es la normalización de la representación de algunos metadatos y el cumplimiento de ciertos metadatos de forma obligatoria, recomendada u opcional. Para ello, en sus respectivas especificaciones se define un formato de metadatos común de interoperabilidad, Dublin Core, sobre el cual se deben plasmar los metadatos de

acuerdo a reglas sintácticas de obligatoriedad, dominio de datos y orden de metadatos. Para refinar aún más el dominio de los datos expuestos, se determinan y utilizan vocabularios específicos (p.e. info:eu-repo/semantics/) o códigos estándares (p.e. ISO 639-3). Esta estandarización en la representación y codificación de metadatos, además de los protocolos de comunicación, permite la interoperabilidad entre los distintos sistemas de información[16].
En la TABLA 1 se enuncian algunas de las diferencias y similitudes entre los aspectos de las directrices analizadas.

	DRIVER 2.0	**SNRD 2012**	**OpenAIRE 1.1**	**OpenAire 3.0**
Definición de set OAI-PMH:	*setName*: Open Access DRIVER set" *setSpec*: driver	*setName*: Sistema Nacional de Repositorios Digitales *setSpec*: snrd	*setName*: EC_fundedresources set *setSpec*: ec_fundedresources	*setName*: OpenAIRE *setSpec*: openaire
Condiciones de acceso al contenido	(a) en acceso abierto (b) con embargo Se recomienda no exponer contenido embargado.	(a) acceso abierto (b) con embargo.	(a) en acceso abierto (b) con embargo, (c) restringidos	(a) en cceso abierto, o (b) resultantes de algún proyecto de investigación financiado, independientemente de su estado de acceso
Uso de <dc:rights>	*Uso*: Recomendado. *Esquema*: no especificado	*Uso*: Obligatorio. *Esquema*: vocabulario OpenAIRE 1.1 para niveles de acceso. Se debe exponer como la primera instancia.	*Uso*: Recomendado. *Esquema*: vocabulario propio de 4 niveles de acceso: closedAccess, embargoedAccess, restrictedAccess, openAccess.	Ídem OpenAIRE 1.1.
Uso de <dc:format>	*Uso*: Recomendado *Esquema*: MIME de IANA.	*Uso*: Obligatorio *Esquema*: MIME de IANA.	Ídem DRIVER.	Ídem DRIVER.
Tipologías y versiones - <dc:type>	*Uso*: 1. Tipo Driver (Obligatorio) 2. Otros tipos (Opcional) 3. Versión de la obra (Opcional) *Esquema*: define 2 vocabularios propios para tipos y versiones en	*Uso*: 1. Tipo Driver (Obligatorio) 2. Tipo SNRD (Obligatorio) 3. Versión de la obra (Opcional) *Esquema*: usa la misma tipología que Driver y define una propia	Ídem a DRIVER.	Ídem a DRIVER.

	info:eu-repo/semantics.	para tipos y obligatoriedad: (1) - TIPO Driver (Obligatorio) (2) - TIPO SNRD (Oblig.) (3) - Versión TIPO Driver (Opcional)		

TABLA 1 - Algunas diferencias y similitudes entre las directrices analizadas.

Resulta conveniente considerar todas éstas cuestiones comunes entre las directrices, para poder determinar qué particularidades se puede reutilizar en las implementaciones de cada contexto.

Una extraña diferencia surge en relación a la inclusión de la fecha de fin embargo. Por un lado SNRD recomienda indicar dicha fecha en la segunda instancia del metadato dc.rights, ej:

<dc:rights>2016-12-31</dc:rights>

Por otro lado, OpenAire recomienda utilizar una instancia del metadato dc.date bajo la sintaxis 'info:eu-repo/date/embargoEnd/AAAA-MM-DD', Ej:

<dc:date>info:eu-repo/date/embargoEnd/2016-12-31</dc:date>

Módulo OAI 2.0

El módulo OAI2.0[1] está desarrollado en base a la librería XOAI Java Toolkit[2] y pensado específicamente para ser usado en repositorios DSpace[9] como data provider OAI-PMH.

Esta aplicación presenta una configuración flexible que permite su adaptación a diversas necesidades, ya sea para cumplir con políticas institucionales, con directrices de interoperabilidad u otros criterios arbitrarios. Su implementación se basa en 4 conceptos fundamentales: Contexto, Filtro, Transformador y Mapeador.

Un *contexto* OAI 2.0 denota una selección arbitraria de registros del repositorio que cumplen con un conjunto de características determinadas. Existe un contexto por defecto denominado *request* que incluye todos los recursos del repositorio.

Los *filtros* determinan qué ítems del repositorio deben ser incluidos en el contexto solicitado. En su implementación, cada filtro sabe 2 cosas: I) qué características deben tener los registros a incluir en el contexto y II) si dado un registro, este debe ser incluido o no en el contexto. Entonces, cada filtro:

I. determina las características de los recursos a preseleccionar a partir de una o más condiciones sobre los metadatos. Esta condición se deberá generar tanto para código de consulta SQL como de consulta Solr. Luego, estas condiciones se utilizarán en combinación con todas las cláusulas de todos los filtros para determinar el conjunto base de recursos que se usará para los siguientes pasos.

II. determina si un ítem es válido en el contexto a partir del análisis de los datos contenido en el ítem, ya sean metadatos descriptivos, administrativos o incluso sus archivos.

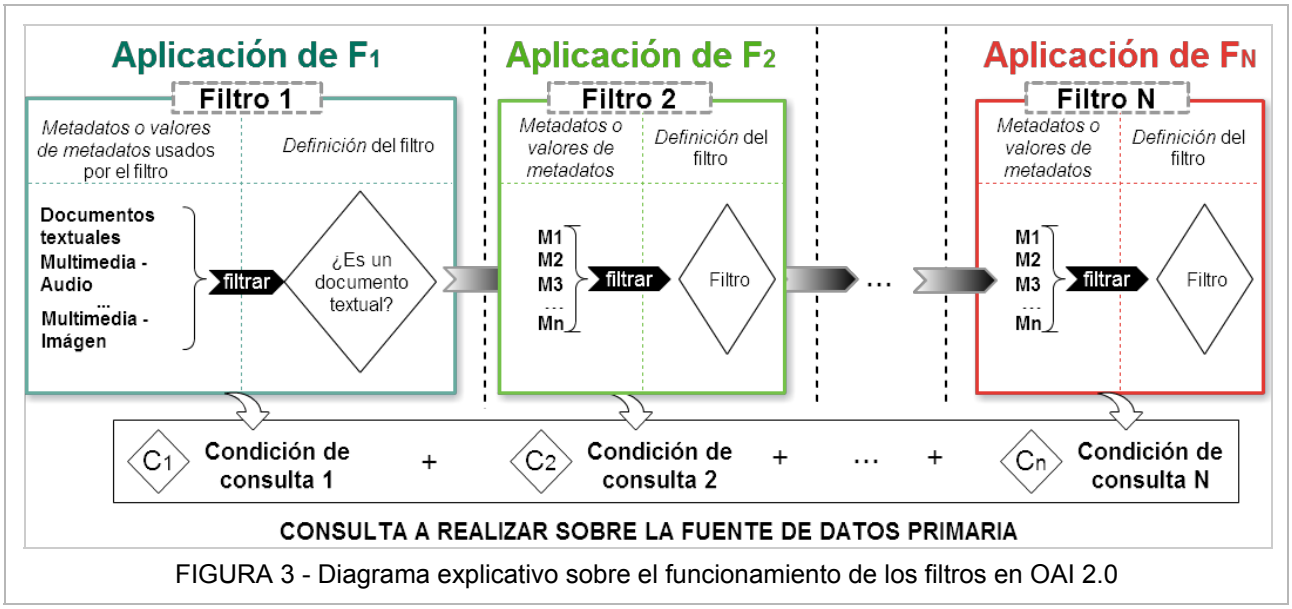

FIGURA 3 - Diagrama explicativo sobre el funcionamiento de los filtros en OAI 2.0

Un *transformador* actúa directamente sobre los metadatos de los documentos filtrados, convirtiendo un conjunto de metadatos de entrada en otro conjunto de metadatos de salida. Una transformación puede implicar tanto una reducción como incremento en los datos de salida, así como también una alteración de valores de dominio.

El *mapeador* toma la salida del transformer y la modifica de acuerdo al formato de metadatos requerido, típicamente oai_dc.

Como se muestra en la TABLA 2, la utilización de estos componentes varía según el verbo OAI-PMH en uso.

¿Qué componentes se utilizan para responder cada uno de las solicitudes OAI-PMH?	Filtro	Transformador	Formato
Identify	No	No	No
ListRecords	Sí	Sí	Sí
GetRecord	Sí	Sí	Sí
ListSets	No	No	No
ListMetadataFormats	No	No	No
ListIdentifiers	Sí	No	No

TABLA 2 - Relación entre verbos OAI-PMH y componentes XOAI

Configuración

DSpace permite personalizar el funcionamiento de OAI 2.0 mediante varios archivos ubicados en la sección de configuración, es decir, en el directorio *config* de la instalación. En la figura 4 se puede ver la estructura de archivos de datos y principales archivos para configurar y personalizar el módulo.

El archivo de propiedades *oai.cfg* especifica las propiedades elementales de configuración del módulo, como ser el tipo de almacenamiento desde la que se procesarán los metadatos de los ítems del repositorio; puede usarse la base de datos o Solr. En este último caso,

debe completarse también la propiedad solr.url con la URL de acceso al índice oai dentro de Solr.

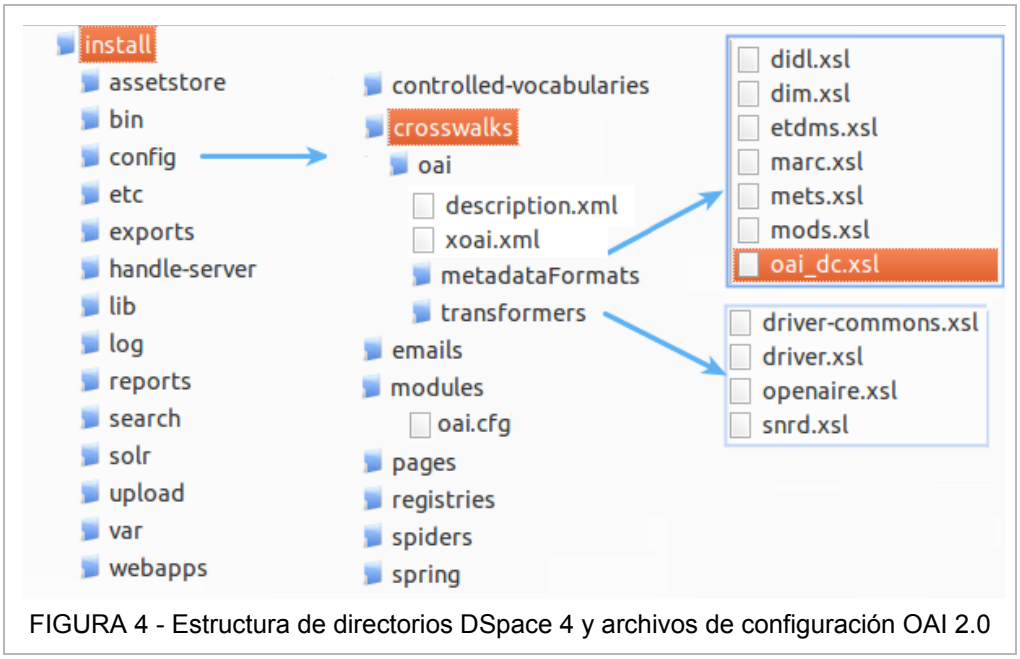

FIGURA 4 - Estructura de directorios DSpace 4 y archivos de configuración OAI 2.0

El archivo de configuración *xoai.xml* es el punto principal para armar y enlazar todos los componentes del módulo, definiendo los contextos habilitados, filtros, transformadores y formatos de metadatos disponibles en cada contexto.
El directorio transformers contiene hojas de estilo de transformación definidas en lenguaje XSLT[10], que suelen usarse para agregar y eliminar metadatos, para alterar su orden o realizar cualquier otra modificación sobre los mismos.
Finalmente, el directorio *metadataTransformers* reúne las hojas de estilo XSLT para mapeo de metadatos a un formato de metadatos específico. Estos archivos son también referenciados desde xoai.xml. A diferencia de los transformers, su función es transformar los el formato interno de metadatos original a un formato de metadatos en particular. El formato deseado se determina en función del parámetro metadataPrefix del de la solicitud OAI-PMH, y de la configuración definida en XOAI.xml para el contexto seleccionado.

Funcionamiento en etapas

El procesamiento de una solicitud OAI-PMH en el módulo OAI 2.0 implica el procesamiento de los componentes enunciados en el inciso anterior. Los componentes puntuales invocados en cada solicitud dependen exclusivamente del contexto seleccionado y de su configuración. A continuación se describe el procedimiento que lleva adelante el módulo para resolver una solicitud de tipo *ListRecords* a partir de la enunciación de 4 etapas sucesivas:

1. Inicio: A partir de la URL solicitada se determina el contexto deseado, el verbo OAI-PMH y un formato de metadatos, entre otros parámetros. Luego, en función de la configuración definida en *xoai.xml*, se determinan los *filter*, *transformers* y *metadataFormats* definidos para el contexto solicitado y que se utilizarán en las siguientes etapas.
2. Selección: Se invoca a cada uno de los filtros determinados en el paso anterior para definir las condiciones lógicas de selección de datos, que luego se combinarán para generar una consulta a la fuente de datos definida —*database o solr*—. Una vez generada dicha consulta, se ejecuta y se obtiene un conjunto de ítems del repositorio.

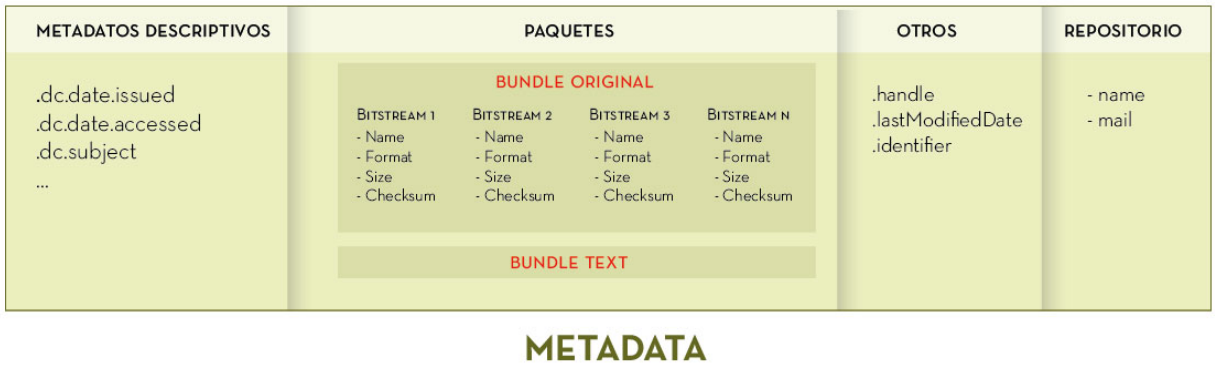

FIGURA 5 - Representación de metadatos XOAI.

3. Transformación: Se procesan los ítems resultantes del paso anterior a partir del transformer definido en el paso 1. Los transformers pueden alterar los metadatos de cualquier forma, aunque deben mantener el formato XOAI[14] (ver Figura 5) de forma que, al continuar con el procesamiento en el paso 4, se conozca la estructura de ítems recibida.
4. Mapeo/crosswalk: esta fase consiste en tomar los ítems resultantes del paso anterior y aplicarles una nueva transformación XSLT para adecuar su estructura, actualmente en formato XOAI, al formato de metadatos solicitado.

Finalizada la última etapa, los metadatos son devueltos en la respuesta OAI-PMH. En caso de recibir una solicitud de verbo *GetRecord*, el procedimiento es prácticamente igual al planteado, con la siguiente excepción en el paso 2: en lugar de construir la consulta de recuperación de datos en función de los filtros, el módulo recupera el ítem y le pide luego a los filtros que lo validen. Si todos los filtros aprueban el ítem se procede con la etapa 3, caso contrario el procesamiento de la solicitud es abortado.

Personalización del módulo OAI 2.0

Cabe destacar que el módulo OAI 2.0 de DSpace ya provee una definición para las directrice DRIVER y OpenAire, consistente en configuraciones genéricas que deben ser adaptadas a la estructura y necesidades de cada repositorio. De acuerdo a las características de los metadatos internos del repositorio será la complejidad y cantidad de filtros necesaria y la complejidad del transformer.

La configuración del *data provider* requiere tomar una serie de decisiones que incluye:

I. Determinar a qué directrices el repositorio debe, o conviene, adecuarse. En general se querrá mantener configuraciones separadas por lo que para cada conjunto de directrices tendremos un contexto.

II. Determinar qué ítems cumplen con las condiciones necesarias planteadas por las directrices y cuáles no. Por ejemplo, si se desean documentos de tipologías específicas o con acceso irrestricto, entre otros. Estas condiciones deberán ser traducidas en filtros que segreguen las obras deseadas por el contexto.

III. Determinar qué requisitos estructurales plantean las directrices y analizar si es posible adecuar los metadatos internos a los mismos. Es normal que en las directrices se requieran metadatos que no existen en los ítems aunque, en general, es posible derivarlos a partir de otros metadatos pre-existentes. Estos cambios y derivaciones se plasman dentro de los transformers.

IV. Determinar el esquema que debe mantener cada metadato de acuerdo a las restricciones de las directrices. Para algunos metadatos, las directrices requieren o aconsejan un esquema o dominio de datos válido. Algunos ejemplos son: iso639-2 para dc.language, tipología DRIVER para el campo dc.type, etc. Sin embargo y dado que internamente el repositorio guarda los metadatos bajo esquemas propios o diferentes al necesario, no queda otra opción que realizar un mapeo de esquemas donde se corresponda cada valor del repositorio con el esperado por las directrices. Al igual que el caso anterior, estos mapeos de esquemas se realizan desde el transformer.

V. Determinar los formatos de metadatos necesarios en la diseminación. A partir de ésta decisión, se definirán los formatos del contexto.

Creación de contextos

En primer lugar se comienza con la declaración de cada contexto en el archivo xoai.xml, creando una etiqueta *Context*, con el atributo *baseurl* indicando el nombre público del contexto. Posteriormente en este elemento *Context*, haremos referencia a los otros componentes: *Filter,Format,Transformer* y *Set.*

Es posible generar un *set* dentro de cada contexto que simplemente incluya todo el contenido del contexto de acuerdo a lo aceptado por sus filtros. Esta práctica es usuario y requerida en las 3 directrices ya mencionadas. En la Figura 6 se incluye un extracto de la definición de un contexto *snrd*, que retorna los recursos del repositorio que cumplen con las restricciones planteadas en las directrices SNRD.

```
<Context baseurl="snrd">
        <Set refid="snrdSet"/>
        <Filter refid="snrddocumentsubtypeFilter"/>
        ...
        <Filter refid="fulltextOnlyFilter"/>
        <Filter refid="existsBundleOriginalFilter"/>
        <Transformer refid="snrdTransformer"/>
        <Formats refid="oaidc"/>
</Context>
```

FIGURA 6 - Declaración de un contexto en OAI 2.0.

Creación de Filtros (filters)

Para la definición de los filtros fue necesario analizar las directrices y detectar en cada caso qué tipologías de recursos eran aceptadas y cuáles eran las condiciones mínimas y obligatorias que debía cumplir. Luego, en cada caso se determinó si era necesario utilizar un filtro o si se podría resolver de otra forma, como por ejemplo desde un transformer.

La cadena de filtros a utilizar para cada contexto se especifica en la configuración del archivo *xoai.xml*, dentro de la declaración del contexto. Se pueden utilizar tanto filtros preexistentes —del paquete *org.dspace.xoai.filter*— como filtros propios, implementados en lenguaje Java © de acuerdo a como se explicará más adelante.

En general, los filtros aceptan algún valor en su configuración, dependiendo de los parámetros que éstos esperan para su funcionamiento. En la TABLA 3 se detallan algunos filtros predefinidos que seguramente se utilizarán en cualquier implementación:

Filtro	Descripción	Parámetros esperados

DSpaceAtLeastOne MetadataFilter	Selecciona a partir del parámetro *field* los ítems que tienen dicho metadato válido de acuerdo los parámetros indicados en *operator* y *values*. Los operadores posibles son: CONTAINS, EQUAL, GREATER, LOWER, STARTS_WITH, GREATER_OR_EQUAL, LOWER_OR_EQUAL, ENDS_WITH. Por ejemplo si instanciamos este filtro con los parámetros field=type, operator=EQUAL y values="Article, Book" tendremos un filtro que acepta solamente artículos y libros.	- field - operator - values / value
DSpaceMetadata ExistsFilter	Selecciona los ítems que contienen al menos una instancia de alguno de los metadatos indicados por el parámetro *fields*.	- field/fields

TABLA 3 - Algunos filtros por defecto en OAI 2.0

En particular, los filtros mencionados en la tabla 3 proveen solamente acceso a los metadatos descriptivos del ítem. En cambio, si se desea filtrar a partir de otro criterio se deberá implementar un filtro ad-hoc, por ejemplo para hacer validaciones sobre bundles, bitstreams o visibilidad, entre otros.

A partir de DSpace 5.0 y OAI 2.1 también será posible generar filtros compuestos definidos como una combinación lógica de otros filtros: AndFilter, OrFilter y NotFilter. Estos filtros lógicos permitirán:

- reusar filtros ya definidos a partir de su combinación o negación
- crear nuevos filtros que seleccionen complementos. Es decir, en lugar de indicar por extensión todo lo que se quiere, se indica qué es lo que no se quiere.

Implementación de filtros nuevos

Cada filtro personalizado en OAI 2.0 debe incluir la lógica de validación estipulada por la clase *org.dspace.xoai.filter.DSpaceFilter*, teniendo en cuenta que cada método de filtrado se aplica sobre un modelo particular y en consecuencia de solicitudes OAI-PMH diferentes (ver Tabla 4). Se presenta entonces una situación muy propensa a errores que debe ser analizada con precaución al momento de configuración, especialmente cuando los filtros validen a partir de los bistreams o de otros datos que NO sean sus metadatos descriptivos.

Verbos OAI-PMH	**Método**	**Descripción**	**Modelo**
ListRecords, ListIdentifiers	getWhere()	Devuelve una condición SQL a utilizar en la consulta sobre la base de datos usada por DSpace.	Base de datos
ListRecords, ListIdentifiers	getQuery()	Devuelve una condición SOLR a utilizar en la consulta sobre el motor de indexación.	Documento Solr
GetRecord	isShown()	Verifica si dado un ítem, el mismo es apto para ser incluido en el contexto.	DSpaceItem

TABLA 4 - Métodos a implementar sobre un filtro propio.

Se resume a continuación las características de cada uno de estos modelos:

1. Base de datos: es el modelo de entidades de DSpace, representado por sus tablas en la base de datos. Provee acceso irrestricto a todos los datos del repositorio.

2. Documento Solr: contiene los datos almacenados en el *core* de Solr correspondiente al móodulo OAI2.0. Cada documento mantiene datos sobre un ítem, que provienen de la tabla item, y sus metadatos descriptivos, que provienen de la tabla *metadatavalue*. Ver Figura 7. También contiene dos campos particulares:
    - *item.compile*: campo de cache que contiene un texto con el modelo XOAI completo del ítem serializado en XML, y
    - *metadata.dc.format.mimetype*: campo derivado que contiene el MimeType del primer bitstream del bundle ORIGINAL.
3. DSpaceItem: es el modelo XOAI mencionado previamente en la Figura 5, que contiene metadatos descriptivos, datos sobre bundles y otros datos sobre el ítem y el repositorio.

FIGURA 7 - Estructura de un documento Solr accesible desde el método *getQuery*

Obviar estas diferencias durante el armado de los filtros puede resultar en que los mismos no funcionen de acuerdo a lo esperado y/o que se comportan distinto en un GetRecords de un ListRecords. En particular

A modo de ejemplo se incluye en la Figura 8 la implementación de la clase *DSpaceExistsBundleOriginalFilter* correspondiente al filtro 2.2 que se explicará posteriormente.

Creación de transformadores (transformers)

Luego de implementar los filtros se define si es necesario definir transformaciones sobre los metadatos. En caso de ser necesario alterar los ítems antes de pasar a la siguiente etapa de mapeo, se debe definir un archivo de transformación XSLT. Ésta hoja de estilos actuará sobre la estructura de metadatos XOAI (ejemplificada en la Figura 5), pudiendo inhabilitar la exportación de ciertos metadatos, transformar el valor de los existentes o generar algunos nuevos.

Para definir un transformer es necesario editar el archivo xoai.xml :

- declarar el nuevo transformer insertando un elemento *Transformer*, el cual consta de un atributo *id* arbitrario y un elemento subordinado *XSLT* que contiene una ruta relativa al archivo de transformación XSLT a utilizar dentro del directorio transformers.
- referenciar desde el contexto deseado al transformer recién declarado a partir de su atributo *id*.

```
public class DspaceExistsBundleOriginalFilter extends DSpaceFilter{

   public SolrFilterResult getQuery(){
      return new SolrFilterResult("metadata.dc.format.mimetype:[* TO *]");
   }

    public DatabaseFilterResult getWhere(Context context){
        try{
            return new DatabaseFilterResult("EXISTS ("+
                "SELECT distinct r.bundle_id "+
                "FROM" +
                    "(SELECT a.bundle_id "+
                    "FROM " +
                    "(SELECT * "+
                        "FROM item it INNER JOIN item2bundle i2b ON it.item_id = i2b.item_id" +
                        "WHERE in_archive AND item_id = i.item_id) AS a " +
                    "INNER JOIN "+
                        "bundle b ON a.bundle_id = b.bundle_id" +
                    "WHERE name = ?) as r " +
                "INNER JOIN "+
                    "bundle2bitstream as b2b on r.bundle_id = b2b.bundle_id" +
                "ORDER BY r.bundle_id"+
            ")", "ORIGINAL");
        }catch (Exception e){
            log.error(e.getMessage(), e);
        }
        return new DatabaseFilterResult();
    }

    public boolean isShown(DSpaceItem item) {
        Iterator<Element> metadata = item.getMetadata().getMetadata().getElement().iterator();
        while(metadata.hasNext()){
            Element metadataElement = metadata.next();
            if (metadataElement.getName().equals("bundles")){
                Iterator<Element> bundles = metadataElement.getElement().iterator();
                while(bundles.hasNext()){
                    Element bundle = bundles.next();
                    if (bundle.getName().equals("bundle")){
                        Iterator<Field> fieldIter = bundle.getField().iterator();
                        while(fieldIter.hasNext()){
                            Field field = fieldIter.next();
                            if(field.getName().equals("name") && field.getValue().equals("ORIGINAL")){
                                return true;
                            }
                        }
                    }
                }
            }
        }
        return false;
    }
}
```

FIGURA 8 - Codificación de métodos para filtro particular.

Creación de mapeadores (metadataFormats)

Ésta es la etapa final realizada por el módulo antes de retornar la respuesta al verbo OAI-PMH *GetRecord* o *ListRecord* recibido. Los archivos utilizados aquí son similares a los

archivos de transformación anteriores pero en lugar de modificar los datos y metadatos, se encargan principalmente de transformar los nombres de los metadatos en base a mapeos hacia un formato específico. Incluso reciben los metadatos bajo la misma estructura XOAI indicada en la Figura 5.
Por cada formato de metadatos declarado en el archivo xoai.xml, existe un archivo XSLT ubicado en el directorio *crosswalks/oai/metadataFormats/* que se encarga de realizar las correspondencias desde el formato interno de metadatos a un formato específico, como por ejemplo oai_dc[11], mets u otros. Cada mapeador de formato puede ser usado en cualquier contexto, agregando el elemento *Format* dentro de la declaración del elemento *Context* de xoai.xml.
En esta etapa se debe prestar particular atención a que los metadatos recibidos sean manejados explícitamente. Cualquier omisión de mapeo provocará que no se exponga algún metadato, afectando de forma negativa la calidad del recurso para los cosechadores.
Típicamente, cada metadato se maneja de las siguientes maneras:

A. se exporta bajo el mismo nombre que trae. Caso típico: se usa internamente *dc.type* y se debe exponer *dc.type*
B. se cambia el nombre del metadato interno por otro del formato destino. Ej. se pasa de *sedici.author.corporate* a *dc.creator*.
C. Se combina 2 o más metadatos en uno solo. Ejemplo: se toma los campos internos *dc.title* y *sedici.subtitle* para generar el campo destino *dc.title* como la unión de los 2 anteriores separados por ': '.
D. Se omite su exportación en caso que el metadato no sea relevante o no se desea que quede público. Ejemplo: *sedici.description.fulltext* es un campo que no debe ser exportado.

En cualquier caso pueden aplicarse transformaciones sobre los datos para respetar un esquema particular, como ISO 8601 para fechas y ISO 639-2 para idiomas, o para normalizar los textos, pasar un metadato a minúsculas, etc.
También resulta importante considerar el orden en que se exportan los metadatos. Por ejemplo, se suele asumir que la primer instancia del campo dc.date es la fecha que representa el recurso por lo que debe exportarse primero la fecha de publicación o presentación. Algo similar sucede con el título principal y títulos alternativos, autores y otros campos.

Experiencia en el repositorio SEDICI - UNLP

El repositorio SEDICI contaba originalmente con un único contexto global *por default*, denominado *request*, que respondía a todas las solicitudes OAI-PMH recibidas. Con el objetivo de dar cumplimiento a las directrices DRIVER, OpenAIRE y en particular SNRD, se configuraron y habilitaron los respectivos contextos.
Dado que los metadatos del repositorio SEDICI son bastante particulares y diferentes del Dublin Core tradicional, resultó necesario configurar varios filtros e implementar algunos otros. Se tiene en consecuencia que la solución encontrada difícilmente sea reusable tal y como está, dado que refleja una realidad muy particular.
Se enumeran a continuación sólo algunas condiciones que debieron ser consideradas sobre los ítems a retornar en cada contexto y que buscan ejemplificar los casos más generales:

(a) debe poseer un metadato *dc.type*
(b) la primer instancia del metadato *dc.type* debe tener una de tipología soportada por las directrices,

(c) la segunda instancia del metadato dc.type debe contener la versión,
(d) debe contener al menos un archivo a texto completo y accesible alojado en el repositorio,
(e) la primer instancia del metadato *dc.rights* debe indicar el nivel de acceso a la obra.

Para (a) no resultó necesario hacer nada dado que siempre se cumple que todo recurso del repositorio posee el metadato *dc.type*.
Para (b) debe tenerse en cuenta 2 cuestiones: b.1) que los tipos de documentos de SEDICI son propios y no coinciden con los de las directrices por lo que resulta necesario hacer una adaptación o mapeo desde los tipos internos a los de la directriz en cuestión, y b.2) que no todos los tipos internos pueden ser adaptados a los de las directrices y que por tanto no deben incluirse en el conjunto resultado. El punto b.1) debe resolverse dentro del transformer mientras que el b.2) requiere la aplicación de un filtro.
El caso (c) presenta una situación diferente ya que requiere la existencia de un metadato que no está presente en el formato interno del repositorio. Sin embargo, resultó posible derivar este campo de *versión* en función del tipo y subtipo de cada documento. Dado que se debía agregar un metadato al ítem, se determinó que era necesario agregar esta lógica al transformer correspondiente.
La accesibilidad de un recurso a texto completo para cada ítem (d) es una condición compleja de evaluar dado que en el repositorio se tienen recursos con archivos cargados en DSpace y recursos con referencias a archivos externos, para referencias a recursos alojados en repositorios externos. Además, se tiene también que en ciertos casos el recurso no esta disponible a texto completo sino que solo se tiene un extracto del mismo. En consecuencia, resulta necesario definir un filtro que analice cada ítem y acepte sólo los recursos que tienen al menos un bitstream en el bundle ORIGINAL y que tienen un metadato que indique que está disponible a texto completo (sedici.description.fulltext en SEDICI)
Finalmente el último caso (e) implica 2 puntos: e.1) que se incluya en un metadato *dc.rights* el nivel de acceso y e.2) que dicha instancia se exponga la primera entre otras instancias de *dc.rights*. Ambas tareas implican la modificación del ítem por lo que deberán ser resueltas desde el transformer como sigue: para e.1) se debe derivar el campo en función de otros metadatos y bitstreams asociados al ítem, para e.2) deberá forzarse la salida de esta instancia antes que las otras instancias de dc.rights.

Filtros propios

De esta forma, podemos determinar la necesidad de 3 filtros (ver Figura 9):

1. Caso b.2: que acepte todos los documentos que tengan tipologías deseadas. Se puede implementar a partir de DSpaceAtLeastOneMetadataFilter
2. Caso d: que acepte sólo documentos con bitstreams en bundle ORIGINAL y metadato sedici.description.fulltext con valor 'true'. Se puede implementar usando 2 filtro en secuencia :
    2.1. un DSpaceAtLeastOneMetadataFilter que acepte documentos con metadato sedici.description.fulltext con valor 'true'
    2.2. un filtro DSpaceExistsBundleOriginalFilter, implementado ad-hoc, que valide la existencia de al menos un bitstream en el bundle ORIGINAL

El primer filtro se aplica directamente sobre el contexto *driver*, así como en *snrd* y *openaire*, ya que validan la mayoría de los aspectos de DRIVER. El segundo y tercer filtro son particularmente necesario en el contexto *snrd*.

```
<Filter id="driverDocumentTypeFilter">
        <Class>org.dspace.xoai.filter.DSpaceAtLeastOneMetadataFilter</Class>
        <Parameter key="field">
                <Value>sedici.subtype</Value>
        </Parameter>
        <Parameter key="operator">
                <Value>equal</Value>
        </Parameter>
        <Parameter key="value">
                <Value>Articulo</Value>
                <Value>Documento de trabajo</Value>
                <Value>Revision</Value>
                ...
        </Parameter>
</Filter>

<Filter id="fulltextOnlyFilter">
        <Class>org.dspace.xoai.filter.DSpaceAtLeastOneMetadataFilter</Class>
        <Parameter key="field">
                <Value>sedici.description.fulltext</Value>
        </Parameter>
        <Parameter key="operator">
                <Value>equal</Value>
        </Parameter>
        <Parameter key="value">
                <Value>true</Value>
        </Parameter>
</Filter>
<Filter id="existsBundleOriginalFilter">
        <Class>ar.edu.unlp.sedici.dspace.xoai.filters.DspaceExistsBundleOriginalFilter</Class>
</Filter>
```

FIGURA 9 - Declaración de filtros en xoai.xml

Transformadores propios

De acuerdo con las restricciones mencionadas debe incluirse en implementación las siguientes 4 transformaciones:

1. Caso b.1) mapear tipología interna de SEDICI a la tipología de DRIVER,
2. Caso c) derivar la versión de cada obra en función del tipo y subtipo de cada documento
3. Forzar que la salida de T1 y T2 sean la primer y segunda instancia del metadato dc.type
4. Caso e.1) derivar el nivel de acceso de cada obra en función de otros metadatos y bitstreams asociados al ítem,
5. Caso e.2) la primer instancia del campo dc.rights debe contener el nivel de acceso según vocabulario OpenAIRE

En la Figura 10 se muestran fragmentos XSL que se corresponden con los transformers T1 y T3. No se incluye T2 y T4 dado que son similares a T1, ni T5 porque es similar a T3. Todas las transformaciones agregan nuevos elementos al árbol de metadatos XOAI, utilizando fragmentos de código XSL de transformación.

```
<!--T1: Se calcula el tipo DRIVER -->
<xsl:variable name="driverType">
<xsl:choose>
   <xsl:when test="$sediciType='Documento de trabajo'">
      info:eu-repo/semantics/workingPaper
   </xsl:when>
   <xsl:when test="$sediciType='Preprint'">
      info:eu-repo/semantics/preprint
   </xsl:when>
   <xsl:when test="$sediciType='Articulo'">
      info:eu-repo/semantics/article
   </xsl:when>
   ...
   <xsl:otherwise>
      info:eu-repo/semantics/other
   </xsl:otherwise>
</xsl:choose>
</xsl:variable>
```

```
<!-- T3: se produce dc.type ordenado -->
<doc:element name="type">
   <doc:element name="es">
      <doc:field name="value">
         <xsl:value-of select="normalize-space($driverType)"/>
      </doc:field>
   </doc:element>

   <doc:element name="es">
      <doc:field name="value">
         <xsl:value-of select="normalize-space($driverVersion)"/>
      </doc:field>
   </doc:element>

   <doc:element name="es">
      <doc:field name="value">
         <xsl:value-of select="normalize-space($sediciType)"/>
      </doc:field>
   </doc:element>
</doc:element>
```

FIGURA 10 - Bloques XSLT de transformadores T1 y T3.

Como se indicó en la Tabla 1, cada directriz tiene su recomendación para el uso del metadato *dc.type*. Sin embargo todas las transformaciones de tipología tienen la misma estructura de mapeo o crosswalk entre vocabularios, es decir, que para cada tipo interno de los define una única correspondencia en el vocabulario destino. En el caso de SEDICI se debió definir para el primer transformer (T1) un mapeo que para cada tipo de SEDICI retorne uno DRIVER.

La versión a generar en el segundo transformer (T2) no forma parte del conjunto de metadatos almacenado sobre cada ítem, por lo que deberá generarse teniendo en cuenta la tipología y otras cualidades del material. Básicamente, es muy similar a lo realizado en el caso anterior, es decir, en función del valor de la tipología se determina la versión.

El tercer transformer (T3) tiene la particularidad de forzar la salida de T1 y T2 como primera y segunda instancia del metadato dc.type, seguida de todas las demás instancias existentes. Esta acción se realiza de forma simple en XSLT como se puede ver en la Figura 9, donde se imprime la primer instancia y luego se fuerza la impresión del resto.

El cuarto caso, T4, deriva el nivel de acceso a un ítem (abierto, cerrado, embargado, restringido) según la existencia de un embargo temporal, de archivos asociados al ítem, del nivel de acceso a los mismos y de la existencia de un metadato específico con una URL de acceso externo.

Finalmente, el último transformer (T5) es similar a T3, y hace que la primer instancia del metadato dc.rights contenga el nivel de acceso calculado en T4.

Formatos propios

Para la definición de los mapeos del repositorio SEDICI se utilizó como base el mapeo que trae consigo DSpace, que maneja los casos típicos de mapeo hacia los formatos más usados como oai_dc, mets y mods, entre otros. Particularmente para los contextos *driver*, *openaire* y *snrd* sólo se ajustó el archivo de mapeo *oai_dc.xsl* dado que dichas directrices requieren solo ese formato.

La personalización de oai_dc.xsl se hizo en función de una tabla de mapeos que define para cada metadato interno la correspondencia en el formato destino, es decir, Dublin Core simple. Como se mencionó previamente, las correspondencias pueden ser simples (1 a 1),

compuestas (n a 1), o nulas (1 a 0), y pueden implicar o no transformaciones sobre los datos (ver Tabla 5).

#	Metadato origen	Metadato destino	Notas sobre el valor
1	dc.type	dc.type	no se altera
2	sedici.creator.corporate	dc.creator	no se altera
3	sedici.creator.person	dc.creator	no se altera
4	dc.title sedici.subtitle	dc.title	se combinan los 2 campos con ': ' . Ver Figura 11
5	dc.title.alternative	dc.title	
6	dc.language	dc.language	Se adecua el esquema de idiomas a ISO 639-1

TABLA 5 - Mapeo de metadatos internos a Dublin Core Simple

```
<!-- dc.title=dc.title: sedici.subtitle -->
<xsl:if select="doc:metadata/doc:element[@name='dc']/doc:element[@name='title']/...">
   <dc:title>
      <xsl:value-of select="doc:metadata/doc:element[@name='dc']/doc:element[@name='title']/..." />
      <xsl:if test="position() = 1">
         <xsl:if test="doc:element[@name='sedici']/doc:element[@name='subtitle']/...">
            <xsl:text> : </xsl:text>
            <xsl:value-of select="doc:element[@name='sedici']/doc:element[@name='subtitle']/..." />
         </xsl:if>
      </xsl:if>
   </dc:title>
</xsl:if>

<!-- dc.title=dc.title.alternative -->
<xsl:for-each
select="doc:metadata/doc:element[@name='dc']/doc:element[@name='title']/doc:element[@name='alternative']...">
   <dc:title><xsl:value-of select="." /></dc:title>
</xsl:for-each>
```

FIGURA 11 - Mapeo para el campo *dc.title*

Un caso particular surge al generar el campo dc.format con el tipo MIME de la obra , dado que este no es un dato que se guarde junto a los metadatos descriptivos sino que el mismo está asociado a los bitstreams del item del bundle ORIGINAL. Por ello, la generación de este campo no implica un mapeo como los anteriores sino que requerirá acceder a otra sección del item recibido en formato XOAI dónde se contiene datos sobre los bundles y bitstreams (ver Figura 12 y Figura 13)

```
<!--  bitstream.format -->
<xsl:for-each select="doc:metadata/doc:element[@name='bundles']/
      doc:element[@name='bundle' and doc:field[@name='name' and text()='ORIGINAL']]/
      doc:element[@name='bitstreams']/doc:element[@name='bitstream' and position()=1]/doc:field[@name='format']">
  <dc:format><xsl:value-of select="." /></dc:format>
</xsl:for-each>
```

FIGURA 12 - Generación del campo dc.format a partir del formato del primer bitstream del Bundle Original.

```
<element name="bundles">
   <element name="bundle">
      <field name="name">ORIGINAL</field>
      <element name="bitstreams">
         <element name="bitstream">
            <field name="name">portada-0001.pdf</field>
            <field name="format">application/pdf</field>
            <field name="size">28198</field>
            <field name="url">http://localhost:8080//bitstream/10915%2F1063/1/bitstream</field>
            <field name="checksum">c4e3648443bda78bc452783ef30f2bdf</field>
            <field name="checksumAlgorithm">MD5</field>
            <field name="sid">1</field>
         </element>
      </element>
      ...
   </element>
   ...
</element>
```

FIGURA 13 - Sección del modelo XOAI con el primer bitstream del Bundle Original.

Conclusiones

Gracias a la flexibilidad provista por el módulo OAI 2.0 es posible generar diferentes vistas de un repositorio, tarea que no era posible previamente y que limitaba las vías de interoperabilidad para repositorios basados en el software DSpace en versiones previas a la 3.

El potencial de uso de este tipo de contextos o vistas personalizados es muy amplio y permite construir y exponer subconjuntos arbitrarios de recursos con diversos fines, como ser:

- cumplir con las restricciones de calidad y formato establecidos por una o más directrices de interoperabilidad
- compartir recursos con otros repositorios de materiales seleccionados. Por ejemplo se podría construir un contexto que exponga todos los recursos de una temática particular para que lo coseche un agregador temático o incluso de una tipología en particular para que los coseche un agregador de Tesis, Revistas o Libros.
- definir un mecanismo interno de control y análisis del repositorio, a partir del reporte en un contexto ad-hoc de los recursos que no cumplen con el nivel de calidad deseado por falta de metadatos obligatorios, archivos, etc.
- generar múltiples servidores OAI para permitir mayor diseminación en repositorios multi-institucionales.

Esta flexibilidad aportada por el módulo trae aparejada una mayor complejidad en la configuración, implementación, definición de mapeos y posibles errores que debe afrontar el encargado de interoperabilidad de un repositorio.

Finalmente, el análisis de las directrices y su contraste con el perfil interno de metadatos del repositorio resultará de vital importancia para determinar los requerimientos de cada caso. Como se indicó previamente, cada repositorio tendrá configuraciones muy particulares y dependientes de su realidad.

Referencias

[1] DSpace 4.x Documentation - OAI 2.0 Server- DuraSpace Wiki. Revisado Julio, 27, 2014 desde https://wiki.duraspace.org/display/DSDOC4x/OAI+2.0+Server

[2] Lyncode XOAI in Github - XOAI Wiki - (Revisado en Julio, 27, 2014). Disponible en: https://github.com/lyncode/xoai/wiki

[3] SEDICI - Repositorio de la Universidad Nacional de La Plata - (Revisado en Julio, 27, 2014). Disponible en: http://sedici.unlp.edu.ar/

[4] OpeAIRE_Guidelines: For_Literature_repositories [Directrices OpenAIRE: Para Repositorios Literarios]. - (Revisado en Julio, 27, 2014) - Disponible en: https://guidelines.openaire.eu/

[5] OpenAIRE Guidelines v.1.1 - (Revisado en Julio, 28, 2014) - Disponible en: https://www.openaire.eu

[6] Directrices DRIVER 2.0: Directrices para proveedores de contenido - Exponiendo recursos textuales mediante OAI-PMH] (Revisado en Julio, 28, 2014) - Disponible en: http://www.driver-support.eu/

[7] Directrices SNRD: Directrices para proveedores de contenido del Sistema Nacional de Repositorios Digitales Ministerio de Ciencia, Tecnología e Innovación Productiva. (Revisado en Julio, 28, 2014) - Disponible en: http://repositorios.mincyt.gob.ar/pdfs/Directrices_SNRD_2012.pdf

[8] The Open Archives Initiative Protocol for Metadata Harvesting (2008) - (Revisado en Julio, 28, 2014) - Disponible en: www.openarchives.org/OAI/openarchivesprotocol.html

[9] Lyncode - XOAI Add-On for DSpace - (Revisado en Julio, 28, 2014) - Disponible en: http://www.lyncode.com/dspace/addons/xoai

[10] XSLT Transformation (Version 1.0) - W3C Recommendation - (Revisado en Julio, 28, 2014) - Disponible en: http://www.w3.org/TR/xslt

[11] OAI_DC XML Schema - (Revisado en Agosto, 10, 2014) - Disponible en: http://www.openarchives.org/OAI/2.0/oai_dc.xsd

[12] DSpace Architecture - (Revisado en Agosto, 8, 2014) - https://wiki.duraspace.org/display/DSDOC4x/Architecture

[13] Usage of Open Access Repository Software: Worldwide - OpenDoar 2014 - Revisado en Agosto, 8, 2014) - Disponible en: http://www.opendoar.org/onechart.php

[14] XOAI XML Schema - Internal Schema - (Revisado en Agosto, 10, 2014) - Disponible en: https://github.com/lyncode/xoai/blob/3.x/schemas/xoai.xsd

[15] El caso de Interoperabilidad para Repositorios de Acceso Abierto - (Revisado en Agosto, 14, 2014) - Disponible en: https://www.coar-repositories.org

[16] Herramienta para la recolección de metadatos bibliográficos mediante el protocolo OAI-PMH - Introducción - Yusniel Hidalgo Delgado - (Revisado en Agosto, 14, 2014) - Disponible en: http://www.researchgate.net/publication/235934068_Herramienta_para_la_recoleccin_de_metadatos_bibliogrficos_mediante_el_protocolo_OAI-PMH