UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Título: Herramienta de manipulación de recursos y sus metadatos aplicada a las tareas de control y mantenimiento en un repositorio digital

Autores: Nicolás Schwab

Director: Dra Marisa R. De Guisti

Codirector: -

Asesor profesional: -

Carrera: Licenciatura en Sistemas

Resumen

Esta tesina describe el trabajo realizado para la implementación de una herramienta cuyo objetivo es mejorar y facilitar las tareas de preservación digital que se llevan a cabo en los repositorios digitales, el desarrollo de esta herramienta se realiza en el marco específico del repositorio CIC-DIGITAL, el cual se encuentra basado en el software DSpace.

En primer lugar se estudia y analiza el marco teórico para formar una idea clara sobre las necesidades existentes en los repositorios sobre la preservación digital. Con la idea formada se comienza con la creación de un Lenguaje de Dominio Específico (DSL) que permita a los distintos administradores del repositorio realizar acciones que faciliten la preservación de los elementos que este alberga. Para dicha creación se definen características que debe cumplir el lenguaje y luego se detalla y justifica paso a paso la creación del mismo. Una vez finalizada la creación del lenguaje se explica el desarrollo de la herramienta que brinda a los administradores acceso a dicho lenguaje junto con su funcionamiento y las clases esenciales que la componen. Para finalizar, se detallan diferentes ejemplos de la herramienta en funcionamiento complementados con imágenes reales.

Palabras Claves

CIC-DIGITAL – Repositorio Institucionales – Preservación digital – DSpace – JSR-341 – DSL –Lenguaje de expresiones – Módulo de expresiones –

Conclusiones

Se obtuvo una herramienta que permite a los administradores de un repositorio realizar acciones de validación, selección y transformación sobre los elementos del mismo. Asimismo el lenguaje que provee la herramienta brinda abstracción del funcionamiento interno de la aplicación y su sintaxis resulta simple, expresiva y sin demasiadas referencias o agregados de programación

Trabajos Realizados

Se creó un lenguaje de expresiones de sintaxis simple que pueda ser utilizado por los administradores del repositorio, es decir usuarios finales sin conocimientos en programación. Luego se creó una herramienta que les permite a dichos administradores, a través del lenguaje mencionado, realizar acciones de validación, selección y transformación sobre los elementos del repositorio.

Trabajos Futuros

Se plantean distintas acciones a realizar para probar la expresividad del lenguaje que provee la herramienta. A su vez se enumeran y justifican diferentes módulos que posee DSpace en los cuales la integración de la herramienta podría ser de gran utilidad. Finalmente se detallan diversas mejoras que puedan realizarse para mejorar la funcionalidad que provee la herramienta en cada uno de sus módulos (validación, selección y transformación).

Fecha de la presentación: Febrero del 2017

Índice

Capítulo 1 | Motivación y Objetivos

Motivación:

Actualmente existe un crecimiento sostenido en la cantidad y calidad de los repositorios digitales, cuyas responsabilidades son almacenar, difundir y preservar contenidos digitales. La preservación se refiere a las acciones orientadas a preservar y dar acceso a largo plazo a los contenidos del repositorio a través del tiempo, es decir mantener su usabilidad y accesibilidad a largo plazo, para lograr este objetivo los repositorios deben contar con distintos mecanismos que les permitan a sus administradores llevar un control y manipular dichos elementos según sea necesario. La situación precedente genera cambios constantes en sus elementos y en sus respectivos metadatos, por lo que es necesario que los repositorios tengan mecanismos de control, de evaluación y transformación para permitir a sus administradores el correcto mantenimiento de sus elementos.

La motivación para esta tesina surgió a partir del trabajo realizado en (Terruzi, 2015) y de experiencias previas obtenidas en los repositorios SEDICI (Servicio de Difusión de la Creación Intelectual) y CIC-Digital, con el objetivo de complementarlas y mejorarlas. Si bien los repositorios mencionados precedentemente cuentan con facilidades para la manipulación de ítems, la realización de tales actividades exige conocimientos informáticos especializados del software sobre el software en que está implementado el repositorio e involucra tiempo de desarrollo.

Trabajos anteriores

El equipo de SEDICI, a lo largo del tiempo, ha logrado proponer, implementar y desarrollar distintas soluciones a los problemas que presenta la preservación digital (De Giusti, 2014). En (De Giusti, Oviedo, Lira & Villarreal, 2013) se menciona que “en la medida que el volumen de recursos aumenta, también aumenta la necesidad de contar con mecanismos de control automático de metadatos y archivos”, en el mismo también se plantea la necesidad de implementar tareas de preservación orientadas al chequeo de integridad y creación de nuevos metadatos con el objetivo de ayudar y mejorar los procesos y la tarea de preservación digital. En (Biblioteca Nacional de Australia, 2003) la *United Nations Educational, Scientific and Cultural Organization* (UNESCO) define como preservación digital a “el conjunto de prácticas de naturaleza política, estratégica y acciones

concretas, destinadas a asegurar la preservación, el acceso y la legibilidad de los objetos digitales a largo plazo" en dicho artículo además se presentan las distintas dificultades concretas al objetivo de la preservación digital. Estos documentos, entre otros, evidencian la necesidad de una herramienta que permita:

- Simplificar los procedimientos y tareas de mantenimiento y preservación de los elementos contenidos dentro de los repositorios digitales.
- Tener la posibilidad de modificar no sólo un ítem de manera independiente sino, por ejemplo, colecciones, o conjunto de ítems con determinada característica en común
- Ejecutar tareas automáticamente y de forma periódica
- Crear metadatos no sólo a un elemento en particular, sino a un conjunto de elementos del repositorio, por ejemplo a un conjunto de Ítems/Colecciones
- Eliminar metadatos no sólo a un elemento en particular, sino a un conjunto de elementos del repositorio, por ejemplo a un conjunto de Ítems/Colecciones
- La necesidad de una tarea que pueda aplicar validaciones y generar reportes sobre los ítems y metadatos que no superen satisfactoriamente la ejecución de sus validadores

En (Terruzzi, Lira, Villarreal & De Giusti, 2014) se menciona la posibilidad de automatizar y resolver algunas de las necesidades planteadas anteriormente. Para esto se utilizó un mecanismo provisto por el software DSpace llamado tareas de curación (*DuraSpace Wiki*, 2015) el cual es automatizable y configurable. También se menciona y queda evidenciado que la creación de una nueva tarea de curación requiere un alto conocimiento en programación y tiempos extensos de configuración, por lo que un administrador del repositorio, el cual no necesariamente posee conocimientos avanzados en programación, encontraría dificultosa la tarea de crear una nueva tarea de curación. Es por estas razones que en el ya referenciado artículo se menciona que las tareas de curación son solo una etapa más que marca el crecimiento del control de la preservación en los repositorios y que el siguiente paso es la creación de una herramienta que permita a sus administradores realizar dichas actividades en poco tiempo y de forma independiente.

Casos de Uso:

A partir del análisis realizado sobre los distintos requerimientos planteados por los administradores del repositorio y por la experiencia que posee el grupo de trabajo de los repositorios CIC-Digital y SEDICI, se establecieron distintas necesidades las cuales se han constituído en los problemas a tratar de resolver (no necesariamente todos) con este trabajo

Consultas de Validación:

Para poder seleccionar los distintos elementos del repositorio en base a determinadas características el paso inicial es poder ejecutar diferentes validaciones sobre los mismos. En este contexto es necesario poder realizar al menos las siguientes validaciones sobre el valor de sus metadatos:

- Validar si el valor del metadato es igual a otro
- Validar si el valor del metadato es distinto a otro
- Si el metadato posee valor numérico, validar si es mayor a un cierto número

- Si el metadato posee valor numérico, validar si es menor a un cierto número

Consultas de Selección:

Para poder llevar a cabo cualquiera de las transformaciones, agregado o eliminación de uno o más metadatos es necesario primero seleccionar el elemento o conjunto de elementos ya sea Ítem, Colección o Comunidad que se desea modificar, para esto es necesaria la posibilidad de listar los distintos elementos del sistema, basados en una o múltiples validaciones. Esta acción se debe poder realizar a través de una interfaz gráfica, haciendo más sencillo su uso a los usuarios finales no informáticos, que por ejemplo no están acostumbrados al uso de la consola de comandos. En este caso la herramienta debe permitir al menos:

- Listar todos los Ítems, colecciones o comunidades del sistema
- Listar los Ítems, colecciones o comunidades que cumplan con una o más validaciones

Consultas de Transformación:

Se debe permitir a los usuarios la posibilidad de modificar, agregar o eliminar los metadatos de distintos elementos del sistema previamente seleccionados, con el objetivo de reemplazar los metadatos erróneos, inexistentes o de formato incorrecto. Para ello la herramienta debe permitir:

- Agregar un metadato a dichos elementos, permitiendo asignar como valor del mismo el valor de otro metadato
- Eliminar un metadato de los elementos
- Reemplazar el valor de un metadato, el nuevo valor puede ser el valor de otro metadato
- Evaluar una expresión regular sobre un metadato y reemplazar la primera o todas las coincidencias por un nuevo valor, dicho valor puede ser el valor de otro contenido

Luego de un análisis sobre los casos de uso quedó clara la necesidad de una herramienta configurable que permitiese seleccionar ciertos elementos del repositorio, generar un reporte sobre dicha selección y que además permitiese ejecutar las transformaciones pertinentes sobre los mismos. Además la herramienta debería poder ser utilizada por los administradores del sistema de forma independiente, es decir sin la ayuda de un programador.Resumiendo, la herramienta debería poder realizar controles de calidad orientados a la preservación, ser los suficientemente poderosa para adaptarse a los distintos casos de uso planteados y a su vez debería poder ser utilizada por usuarios sin grandes conocimientos en la programación.

El desarrollo en cuestión debía permitir a los administradores de un repositorio manipular los elementos y sus metadatos sin la necesidad de tener conocimientos avanzados de lenguajes de programación–posibilitando simplificar y agilizar su trabajo al momento de realizar tareas de mantenimiento sobre los elementos del repositorio, ya sea sobre un elemento en particular o sobre un subconjunto de elementos.

En base a los casos de uso planteados y los requisitos que la herramienta debe cumplir se define el objetivo principal de esta tesina y los objetivos secundarios.

Objetivo:

De acuerdo al análisis de los apartados previos, se estableció como primer objetivo de esta tesina desarrollar una herramienta que permita manipular los recursos de un repositorio dado y sus metadatos asociados, a través de un lenguaje específico de uso simple y adaptado al modelo de DSpace, software que se utiliza para gestionar los repositorios donde se pretende utilizar este desarrollo, sin la necesidad de acudir a un programador o a un gestor de base de datos.

Además se plantean, como objetivos secundarios:

- Permitir operaciones de validación, selección y modificación sobre los elementos y sus metadatos.
- Integrar la herramienta en los repositorios SEDICI y CIC-Digital.
- Realizar un pull request al software DSpace con intención de que este sea aceptado e integrado, lo que permitiría que la herramienta sea usable por cualquier repositorio que tome a DSpace como software base.
- Usar el lenguaje específico de dominio aquí propuesto de forma ortogonal en diversos módulos del sistema como son tareas de curación, estadísticas, búsqueda(discovery)(*DuraSpace Wiki*, 2015) y OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) (*DuraSpace Wiki*, 2015), entre otros.

Capítulo 2 | Repositorio Digital

Introducción

Como fue expresado en el capítulo precedente, uno de los objetivos de esta tesina es integrar la herramienta creada en los repositorios digitales institucionales SEDICI y CIC-Digital. En las dos primeras secciones de este capítulo se realiza una pequeña definición de los conceptos repositorios digitales y de repositorios institucionales con el objetivo de poner definiciones de preservación para luego dar una propia y explicar la importancia de la misma en el ámbito de los repositorios digitales.

Definición

Un repositorio digital puede definirse como un sitio web destinado al almacenamiento, recuperación y búsqueda de recursos, un recurso es una representación de un objeto “real” conformado por un conjunto de metadatos que lo describen y brindan información acerca de él, es a través de los metadatos que podemos generar una síntesis y representación del objeto “real”, lo cual nos permite acceder, distribuir y difundir el contenido del recurso sin tener el objeto en sí, sino una representación del mismo. El objetivo de un repositorio es gestionar esos recursos para que puedan ser recopilados, catalogados, difundidos y preservados de forma libre y gratuita, los repositorios permiten que grandes cantidades de información se en contexto al lector sobre el ambiente en el cual se desarrolla este trabajo y la actualidad de los repositorios digitales. Asimismo en la última sección de este capítulo se citan distintas encuentre centralizada, permitiendo un fácil y gratuito acceso de la información a la comunidad. Existen distintos tipos de repositorios digitales entre los que podemos encontrar temáticos, de datos, huérfanos e institucionales entre otros, particularmente haremos hincapié en los repositorios institucionales.

Repositorio Institucional

Un repositorio institucional es un repositorio donde se depositan recursos derivados de la producción científica y académica de de una institución o cualesquiera otros que la institución considera importantes. Sus principales objetivos son facilitar el acceso a estas producciones, aumentar la visibilidad de la producción científica de la institución, asimismo el de preservar los recursos almacenados para asegurar su accesibilidad y uso a largo plazo.

Actualidad de los repositorios digitales

Como fue expresado anteriormente existe un incremento sostenido en la cantidad de los repositorios digitales, esto se debe en parte a la tendencia mundial de exponer la producción de las distintas instituciones académicas a través de repositorios de acceso abierto. Este hecho puede observarse en la figura 2.1, en la cual se muestra un gráfico con el crecimiento en la cantidad de repositorios digitales. Actualmente se pueden encontrar mas de 3200 repositorios en todo el mundo (ver figura 2.2) siendo Estados Unidos el país con más repositorios a nivel mundial.

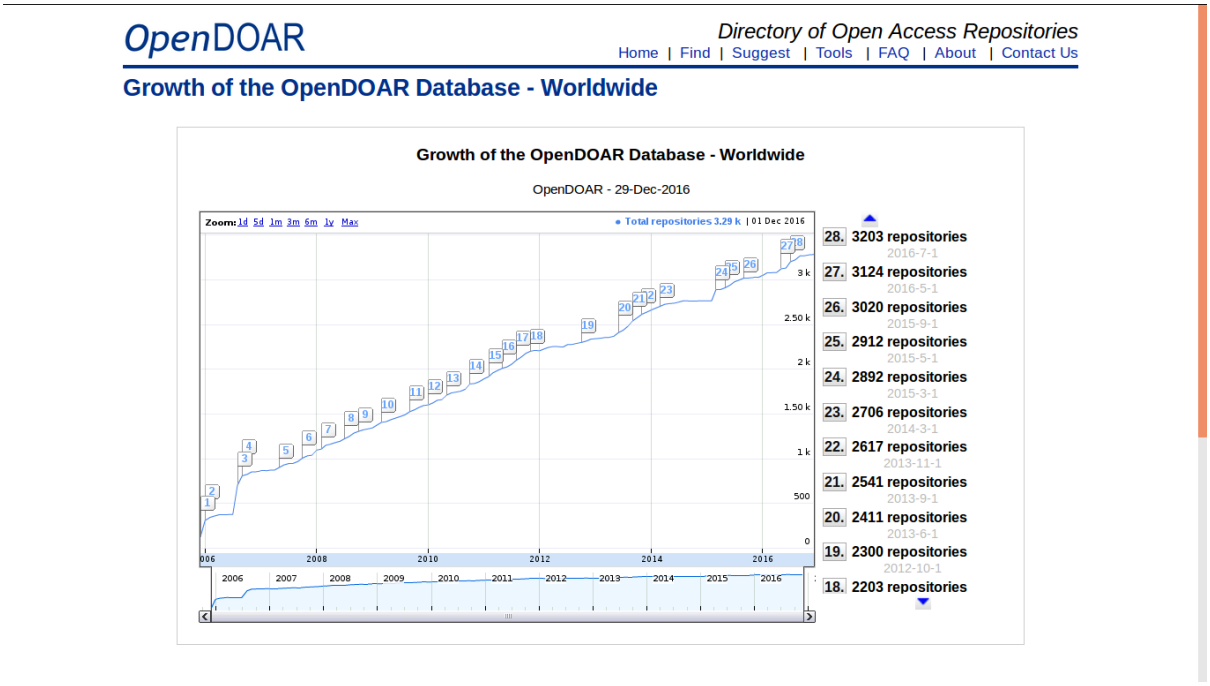


Figura 2.1: Crecimiento en la cantidad de repositorios desde el 2006 hasta la actualidad (OpenDoar)

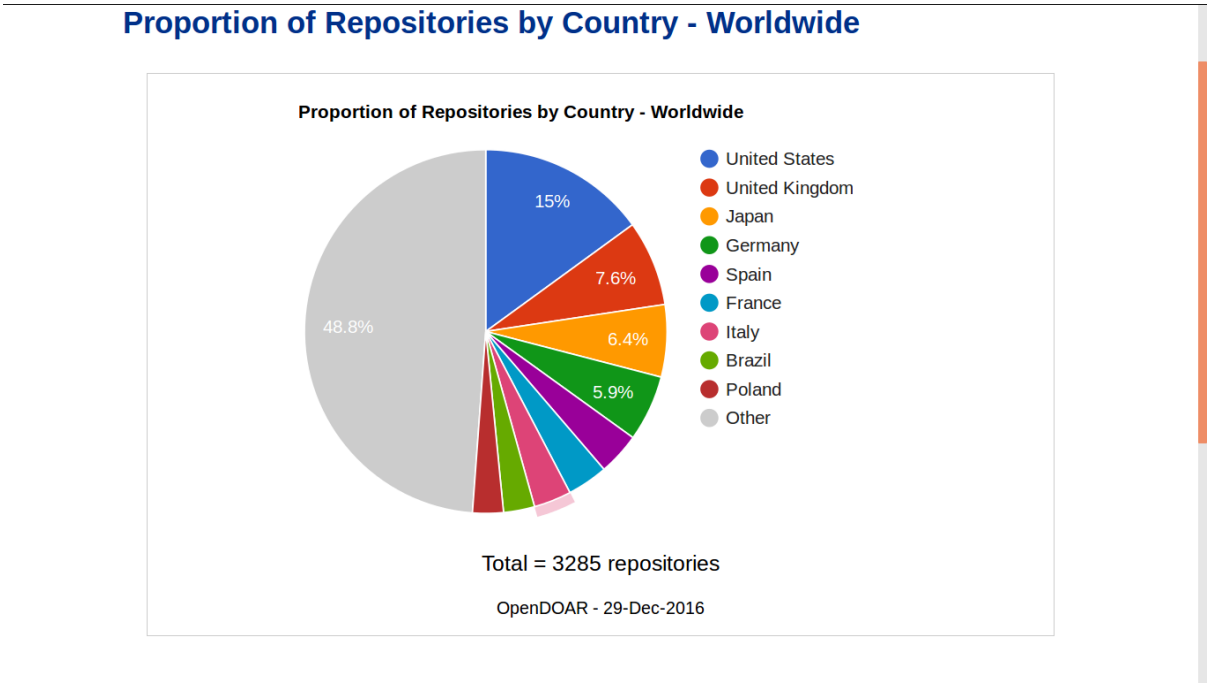


Figura 2.2: Distribución actual de los repositorios (OpenDoar)

Preservación

En los últimos años se generó un crecimiento sostenido en la producción de documentos en formato digital a tal punto que hoy en día la mayoría de las producciones nacen en este formato, dada la dependencia tecnológica de los objetos digitales, este número creciente de contenidos determina la necesidad de actividades para preservar y mantener el acceso a ellos a lo largo del tiempo. Esta preocupación se ve reflejada en el incremento de los distintos trabajos e investigaciones realizadas sobre preservación digital por autores especializados en el área tales como (De Giusti, 2015) y (Strodl, Becker, Neumayer & Rauber 2007).
Para entender mejor la preocupación planteada se debe definir la preservación:

En “Directrices para la preservación digital” (UNESCO 2003) se define la preservación “como las acciones destinadas a mantener la accesibilidad de los objetos digitales a largo plazo”.

En “Preservación digital de documentos. Archivos, bibliotecas y museos” (De Giusti & Villarreal & Plangger, 2015) se define a la preservación como “un conjunto de prácticas de naturaleza política, estratégica y acciones concretas, destinadas a asegurar el acceso a los objetos digitales a largo plazo ”.

Se puede observar en las distintas definiciones de los autores que la preservación digital siempre va encaminada a asegurar el acceso de los objetos digitales a largo plazo, por lo que empieza a recibir importancia tomar acciones o tener herramientas que contribuyan a la preservación digital. Estas acciones o herramientas deben ayudar a resolver los distintos problemas técnicos que se presentan en cuanto a la preservación de los objetos digitales:

- Objetos digitales efímeros dada su mediación tecnológica

- Configuración deficiente del Software
- Control del material
- Protección de la integridad de los datos

Además de los problemas técnicos planteados existen problemas más allá de lo tecnológico, en “Preservación digital en repositorios institucionales GREDOS”(Arévalo, 2011) se categorizan estos problemas en:

- Legales: si no se cuentacon el permiso del titular no podremos preservar un recurso reproduciéndolo o reformateándolo.
- Económicos: si no contamos con los medios necesarios no podremos garantizar la perdurabilidad de los documentos a lo largo de los años.
- Institucionales: si no se asegura el compromiso institucional permanente, si no están convencidos todos los implicados de la necesidad de colaborar, si verificar si se cumple todo lo que se promete, no se podrá preservar a largo plazo.

Como puede verse existen diversos problemas de diferente índole que terminan generando dificultades o imposibilidades para la preservación digital, algunos por ejemplo son legales, es decir si no se tiene un permiso del autor para transformar/migrar el objeto digital éste quedará desactualizado, por otro lado un problema de índole técnica es la naturaleza del objeto digital, ya que al estar mediado por la tecnología si ésta se vuelve obsoleta, el objeto digital se vuelve inaccesible o no legible. En el marco de esta tesina se hará foco en los problemas de índole técnica.

Como fue explicado previamente para resolver los problemas técnicos que presenta la preservación digital, es necesario realizar acciones y contar con herramientas que contribuyan a atenuarlos. En este contexto se puede plantear dos acciones importantes a realizar a la hora de preservar objetos digitales: “Control de calidad e integridad” y “Actividades de saneamiento”. La acciones de control de calidad e integridad sobre los distintos recursos del repositorio identifican el estado de dicho recurso, permitiendo detectar valores erróneos, inexistentes o de formato incorrecto en los metadato, mientras que las actividades de saneamiento permiten corregir dichos errores y devolver la calidad e integridad al recurso. Ambas acciones pueden ser realizadas de forma manual por un usuario con conocimientos técnicos en el área de repositorios, pero realizar estas tareas es un trabajo repetitivo, de larga duración y dado que es realizada manualmente es propensa a errores. Es por eso que es esencial contar con herramientas que permitan realizar estas actividades de forma rápida y semi-automática, para poder reducir la intervención humana lo más posible y así poder minimizar errores. Por todo lo dicho hasta ahora resulta esencial la correcta elección de la herramienta de gestión del repositorio, evaluando las facilidades provistas por dicha herramienta junto con sus agravantes. Refiriendose especificamente al repositorio CIC-DIGITAL, siendo este el repositorio donde se desarrolla el tema principal de esta tesina, fue elegida la herramienta DSpace la cual se detalla en el próximo capítulo.

Capítulo 3 | DSpace

Introducción

Como fue mencionado en los capítulos precedentes en la actualidad existe un crecimiento sostenido en la creación de repositorios digitales, la gestión de los mismos

presenta diferentes problemáticas a tener en cuenta al momento de elegir una herramienta de gestión~~, entre ellas encontramos:~~

- Preservación
- Gestión de metadatos
- Visibilidad de los elementos
- Búsqueda de los elementos
- Gestión de los autores

DSpace (Dspace.org. 2017) es un software de código abierto pensado para la gestión de repositorios digitales que proporciona distintas herramientas y funcionalidades que permiten satisfacer las diferentes necesidades planteadas anteriormente. En la actualidad cerca de la mitad de los repositorios del mundo están desarrollados sobre DSpace (ver figura 3.1), esto se debe a sus distintas características y funcionalidades que lo distinguen entre las herramientas pensadas para la misma finalidad, entre estas características podemos encontrar:

- Posee una gran comunidad de usuarios y programadores a nivel mundial que lo mejoran y actualizan constantemente
- Es un software open source disponible gratuitamente para cualquier persona. El código utiliza la licencia BSD (Berkeley Software Distribution) (Manual, U. P. S. 1984) la cual permite a cualquier institución usarlo, modificarlo y agregar sus propias modificaciones al código.
- Completamente customizable a las necesidades de cada institución, cada una de ellas puede personalizar la forma en la que se verá su repositorio, el formato de metadato que utilizará, la forma y los campos de búsqueda que permitirá, entre otros.
- Puede almacenar y manejar todo tipo de contenido digital. DSpace puede reconocer y manejar distintos tipos de formatos de archivos y Mime Types (*DuraSpace Wiki*. 2015)

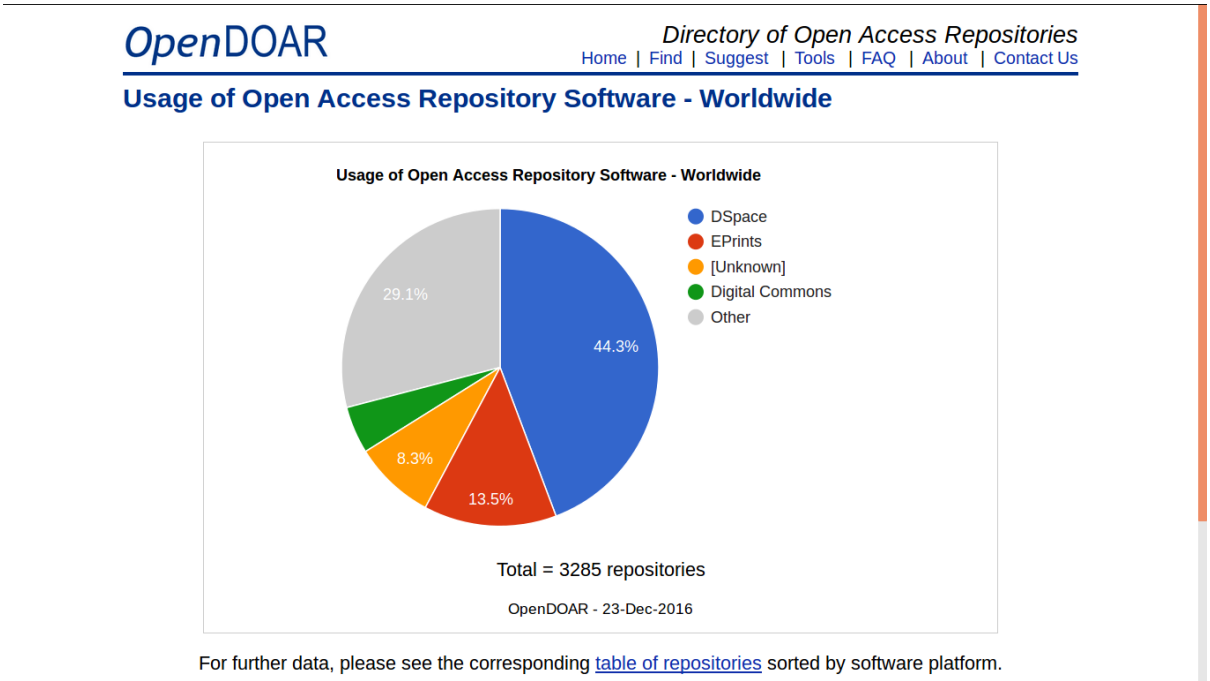


Figura 3.1: Se muestra los software de repositorios más usados con su correspondiente porcentaje según openDOAR

Descripción funcional

Como fue explicado en el subcapítulo anterior, DSpace es el software para gestión de repositorios más usado a nivel mundial, esto se debe a las distintas características y funcionalidades que provee. En este pequeño apartado se hará un pequeña mención y explicación de las funcionalidades más importantes de DSpace.

La funcionalidad principal de DSpace es proveer una presentación organizada de toda la información que posea el repositorio, la misma se organiza en un árbol de comunidades, colecciones e ítems el cual será explicado más adelante con más detalle (ver capítulo Marco teórico subcapítulo DSpace-Estructura). Los Ítems están compuestos principalmente por un conjunto de metadatos, los cuales lo describen (ver capítulo Marco teórico subcapítulo DSpace-Metadatos) y por uno o más archivos, estos archivos normalmente son referenciados como “Bitstreams” dado que una vez que son subidos al repositorios son almacenados como una secuencia (stream) de bits (bit). Otra de las principales funcionalidades que provee DSpace es la de permitir al usuario encontrar la información deseada de manera rápida y sencilla, para lograr esto el software provee buscar contenido a través de los metadatos de los Ítems o por el contenido de los Bitstream (full-text), para complementar este método de búsqueda DSpace tambień permite la navegación por todo el árbol de Comunidades, Colecciones e Ítems y además permite la referencia externa a los elementos del repositorio a través de un identificador persistentes como Handle (Handle, 2017). Otra de las principales características que provee DSpace es que está optimizado para la indexación de Google, esto permite que el contenido alojado en el repositorio sea indexado por Google Search y Google Scholar, esta característica resulta muy importante dado que, según la documentación de DSpace “Repositorios populares soportados por DSpace obtienen un 60% de sus visitas de las páginas de Google” (*DuraSpace Wiki*, 2017).

Metadatos

Los metadatos pueden definirse como datos que describen datos estos guardan información acerca de un elemento, es DSpace todos los elementos contienen metadatos ya sea Comunidad, Colección o Ítem, esto permite almacenar información detallada sobre cualquier elemento alojado en el repositorio. Los metadatos están divididos en tres categorías: Descriptivos, Administrativos y Estructurales. Cada una de las categorías contiene diferente información acerca del elemento, los metadatos Descriptivos almacenan información que permite describir y recuperar el elemento, por ejemplo el título de una tesis. Los metadatos Administrativos contienen información para gestionar el recurso es decir información sobre el mantenimiento del mismo o gestión de derechos entre otros, mientras que los metadatos Estructurales contienen información sobre la estructura interna de los elementos y cómo presentarlo al usuario.

Modelo

El modelo de Dspace consta de 6 elementos: Comunidad, Colección, Item, Bundle, Bitstream y Bitstream Format. Todo repositorio soportado por DSpace está dividido en Comunidades las cuales están compuestas por sub-Comunidades y/o Colecciones, el hecho de que una Comunidad pueda tener sub-Comunidades permite representar la jerarquía que existe en las instituciones, por ejemplo se puede tener una Comunidad llamada Universidad con una sub-Comunidad llamada Facultades. Como ya dijimos las Comunidades también pueden tener Colecciones, estas pueden aparecer en una o más Comunidades lo que permite al igual que las sub-Comunidades, representar mejor la jerarquía de las instituciones y a su vez permite no repetir información, las Colecciones son un agrupamiento de contenido relacionado el cual es representado por el objeto central del modelo, el Ítem. Un Ítem es la representación de un elemento subido al repositorio, este tiene una sola Colección padre sin embargo puede ser referenciado por distintas Colecciones. Cada Ítem posee un conjunto de Bundles, los cuales son agrupaciones de archivos (Bitstream) permitiendo que DSpace maneje a cada Bundle y a los archivos dentro de él de manera diferente. Generalmente un Ítem tiene alguno de los siguiente tipos de Bundle:

- Original: el bundle con el bitstream original depositado: el archivo que el autor o la administración han subido al repositorio.
- Texto: texto completo extraído del bitstream original, correspondiente al Ítem.
- Licencia: contiene la licencia que se subió en conjunto con el Ítem al repositorio, especifica los derechos que se tienen sobre el mismo.
- Licencia CC: contiene la licencia de uso que especifica lo que el usuario final puede hacer con el ítem en cuestión, al tener acceso al mismo.

Un Bitstream es, como el nombre lo indica, una secuencia de bits que representa al archivo subido por el usuario. Ya que dentro del repositorio son almacenados como una secuencia de bits. Cada Bitstream está asociado con un Bitstream Format, esto permite a DSpace realizar tareas de preservación específicas sobre cada formato de Bitstream. En la figura 3.2 se muestra un esquema del modelo de DSpace sacado de la documentación (*DuraSpace Wiki*, 2017).

Es importante aclarar que los objetos Comunidad, Colección e Ítem extienden de una misma clase llamada DSpaceObject (*DuraSpace Wiki*, 2017), los cuales tienen metadatos que los describen (explicados en la sección anterior).

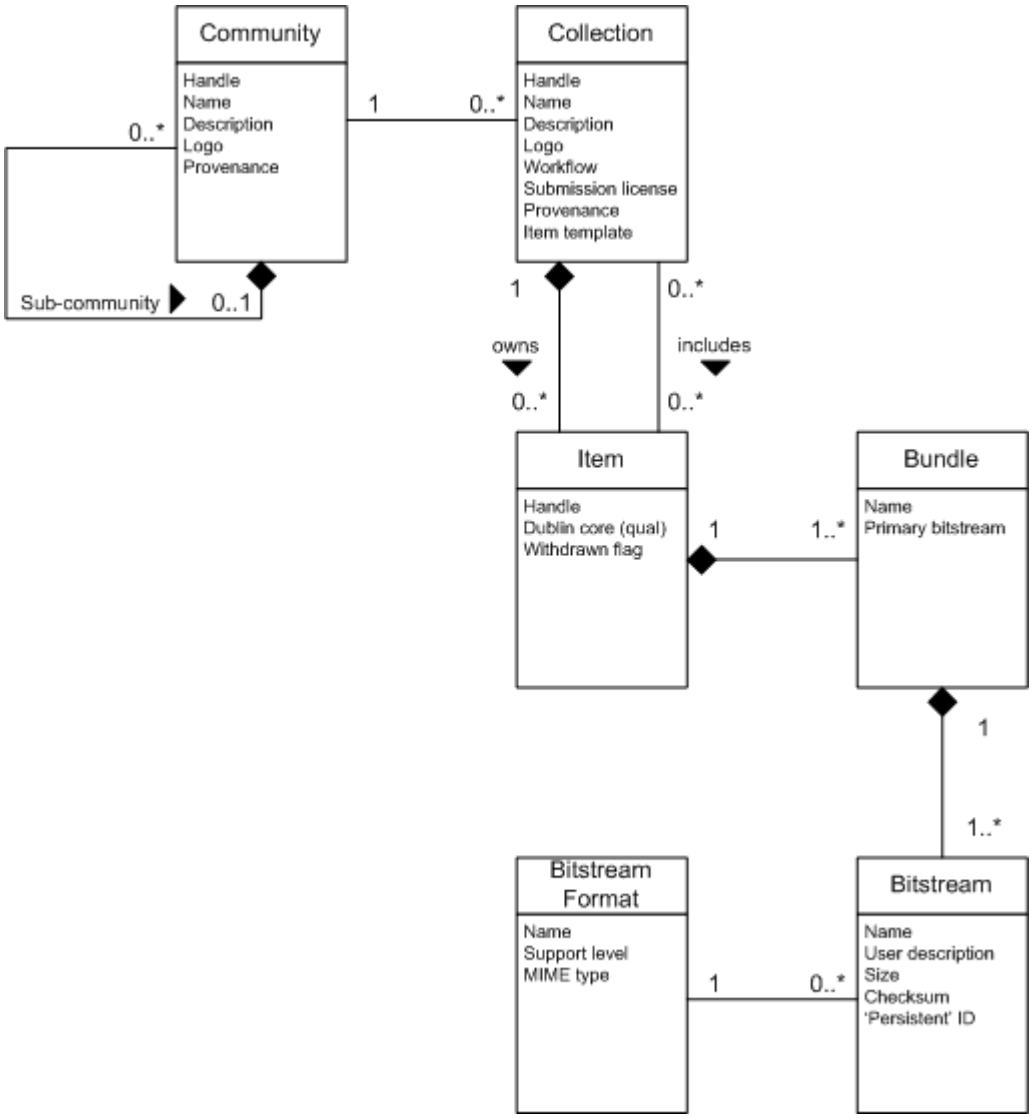


Figura 3.2: Se muestra un esquema del modelo de DSpace

Módulo Additions

DSpace está formado por distintos módulos cada uno creado para cumplir con una funcionalidad específica, en particular el módulo “Additions” fue creado para satisfacer la necesidad de la comunidad de tener un espacio donde pueda agregar o sobreescribir código sin modificar el módulo principal de DSpace (módulo API) (*DuraSpace Wiki*, 2017). Encapsular todas las modificaciones que se quieran realizar sobre módulo API en este módulo es recomendable para que estén contenidos en un único módulo, además esto facilita mucho, la posibilidad de obtener una visión global y un registro de los cambios realizados al software.

Por lo explicado en el párrafo precedente el desarrollo de la herramienta planteada en esta tesina se realizará sobre el módulo Additions.

Tareas de curación

DSpace provee un mecanismo para ejecutar tareas de preservación (ver Trabajos anteriores) el cual llama “Curation Task “(Tareas de Curación) (*DuraSpace Wiki*, 2017), estas tareas pueden operar sobre cualquier DSpaceObject, es decir Comunidad, Colección o Ítem (ver DSpace-Modelo). DSpace provee un número pequeño de tareas de curación por defecto, pero este módulo está pensado para ser extensible es decir que los distintos repositorios puedan generar su propias tareas de acuerdo a sus necesidades. Las tareas de curación tienen acceso y pueden modificar cualquier DSpaceObject, por lo tanto la ejecución de las mismas es una tarea pensada para los administradores del sistema y no para los usuarios finales; las tareas pueden ser escritas en cualquier paquete java de DSpace.

Creación de una nueva tarea

Las tareas de curación no son más que código java, por lo tanto para poder crear una nueva tarea es necesario tener conocimientos sobre dicho lenguaje de programación, además la clase creada debe cumplir con las siguiente características:

- Tener un constructor sin parámetros
- Debe implementar la interfaz "org.dspace.curate.CurationTask"

Activiación

Para poder ejecutar una tarea de curación se debe tener el código correspondiente en algún paquete java y además la tarea debe estar declarada en un archivo de configuración llamado ‘[dspace]/config/modules/curate.cfg’, en el cual se le dará un nombre de referencia. Para esto existe una sección en dicho archivo la cual está preparada para la declaración de todas las tareas de curación:

plugin.named.org.dspace.curate.CurationTask = \
org.dspace.ctask.general.RequiredMetadata = requiredmetadata

Como puede verse dentro del archivo “curate.cfg” existe una propiedad llamada “*plugin.named.org.dspace.curate.CurationTask”,* en dicha propiedad se deben declarar todas las tareas de curación junto con su respectivo nombre de referencia, en el ejemplo dado se está declarando la tarea de curación “*org.dspace.ctask.general.RequiredMetadata*” y se le está dando como nombre de referencia “*requiredmetadata*”. Para agregar una nueva tarea basta con agregar una “,” (coma) al final de una declaración y agregar la nueva tarea de curación.

Ejecución

Las tareas de curación pueden ser ejecutadas a través de la línea de comandos o a través de una interfaz gráfica. Para ejecutar por ejemplo una tarea llamada “vscan” a través de la línea de comandos es necesario que el administrador abra la terminal, se mueva al directorio “[dspace]/bin/” y en el ejecutar el siguiente comando “./dspace curate -t vscan -i

123456789/4”. Si quisiera ejecutar la misma tarea a través de la interfaz gráfica debe ir a alguna de las siguientes páginas del sistema:

- Edición de un DSpaceObject e ir a la pestaña llamada “Curate”
- Existe una página llamada "Curation Tasks" exclusiva para los administradores

Cualquiera de las dos páginas mencionadas presenta un listado con las tareas habilitadas y un botón para ejecutar la tarea seleccionada (ver figura 3.3).

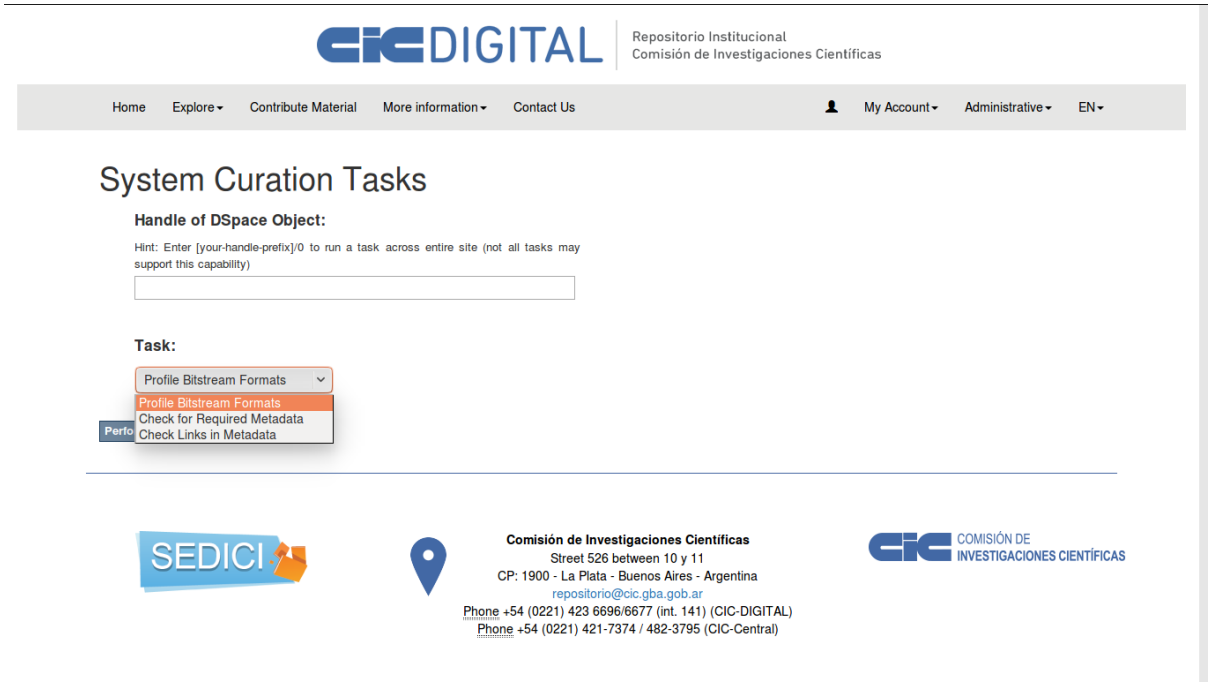


Figura 3.3: Se muestra la pantalla que provee el repositorio CIC-Digital para ejecutar tareas de curación

Por lo explicado en los párrafos precedentes queda evidenciado que la modalidad actual que provee Dspace para realizar tareas de preservación es limitada, porque como se comentó precedentemente para realizarla un administrador para crear una nueva tarea de curación debe tener conocimientos avanzados en en el lenguaje de programación Java y realizar cierta configuración para activar la misma. Es por las razones explicadas que surgió la necesidad de tener una herramienta que permita crear y ejecutar tareas de preservación de forma rápida, simple y que no requiera grandes conocimientos en programación.

Capítulo 4 | JSR-341

Introducción

JSR-341 (Java Specifications Request) es una especificación del lenguaje de programación Java que provee un lenguaje de expresiones (EL), el cual puede definirse como un lenguaje de script para acceder y manipular objetos, en este caso específico el lenguaje de expresiones provisto por la JSR-341 permite acceder y manipular objetos de Java. Como se detalla en JSR-000341 Expression Language 3.0 - Final Release (Jcp.org, 2017) los lenguajes de expresiones fueron introducidos por primera vez en la JSTL(JSP

Standard Tag Library) en su versión 1.0, para permitir a los desarrolladores de páginas web acceder y manipular información de la aplicación sin la necesidad de que conozcan la complejidad asociada al lenguaje de programación Java. Debido a su éxito, tanto la JSP (Java Server Pages) como la JSF (Java Server Faces) desarrollaron cada uno su propio lenguaje de expresión adaptandolos a sus necesidades. Cada uno de los lenguajes de expresiones desarrollados funcionaba únicamente sobre una tecnología específica, es decir que el EL desarrollado para la JSTL funcionaba solo sobre la JSTL, y el desarrollado para la JSP funcionaba solo sobre la JSP y lo mismo ocurría con el EL desarrollado para la JSF, es por esta razón que era evidente la necesidad de tener un solo y unificado lenguaje de expresiones que cumpla los requerimientos y características de cada lenguaje, además de poseer lo mejor de cada uno de ellos. Para lograr esto los grupos de desarrolladores expertos de las tres tecnologías (JSTL, JSP y JSF) crearon la JSR-341, la cual provee un lenguaje de expresiones independiente de la tecnología utilizada en la capa de presentación, es decir que sea utilizable en cualquiera de las 3 especificaciones previamente mencionadas.

Características

Como fue mencionado anteriormente la JSR-341 provee el primer lenguaje de expresiones independiente de la tecnología usada para la creación de la capa de presentación, su objetivo es brindar un fácil acceso a los objetos utilizados por el lenguaje de programación Java y una abstracción a la implementación de los mismos. Para lograr esto el lenguaje de expresiones brinda las siguientes funcionalidades:

- Sintaxis simple y restringida para evaluar expresiones
- Acceso a variables y propiedades anidadas
- Operadores relacionales, lógicos, aritméticos, condicionales y vacíos
- Acceso a funciones implementadas como métodos estáticos en Java

En (Terruzi, 2015) fueron analizadas detalladamente las distintas funcionalidades que provee el lenguaje de expresiones mencionado y se creó un prototipo del lenguaje desarrollado en esta tesina. Dicho prototipo utiliza el lenguaje de expresiones provisto por la JSR-341 y su sintaxis restringida para acceder directamente a los objetos Java, esto ~~le~~ permite realizar una amplia gama de acciones sobre los objetos siendo algunas de ellas validar que un Ítem tenga un identificador persistente, verificar que un artículo posea autor y mostrar todos los Ítems de una Colección que tengan una fecha de publicación específica, entre otros. No obstante acceder directamente a los objetos del modelo de DSpace puede llevar a problemas de seguridad, por ejemplo si el objeto Java Ítem tiene una función que elimina todos sus metadatos esta función podría ser ejecutada utilizando el prototipo mencionado. Para evitar esto, el prototipo utilizaba objetos llamados “Wrappers” los cuales son los que el usuario accede en lugar de los objetos del modelo de DSpace, la finalidad de los wrappers es aplicar un filtro sobre la funcionalidad que provee el objeto del modelo de DSpace para restringir las acciones que puede realizar el usuario. Esta implementación si bien soluciona el problema planteado es propensa a errores, dado que ante una equivocación en la implementación del wrapper se le puede dar acceso al usuario a acciones no deseadas, esto se debe a que la medida de seguridad implementada por el wrapper no es más que un filtro sobre la funcionalidad que provee el objeto según las acciones que se le quieren permiten al usuario.

En conjunto con lo expresado en el párrafo anterior, el prototipo creado en (Terruzi, 2015) obliga al usuario final a aprender y entender la sintaxis del lenguaje de expresiones que provee la JSR-341, esto se puede ver evidenciado por ejemplo en el capítulo 6 sub sección “Desarrollo del módulo de expresiones para DSpace” en el cual se detallan distintas consultas que pueden hacerse con el prototipo, siendo una de ellas:

“Verificar que un artículo posea autor”:

- item.getMetadata.stream().anyMatch(m->m.name == “dc.creator”)

Este ejemplo evidencia lo planteado en el parrafo anterior, dado que para que un usuario puede escribir esta consulta no le hace falta solamente con conocer los objetos del dominio de DSpace, sino que también debe conocer la sintaxis del lenguaje de expresiones de la JSR-341 como lo es “m->m.name == “dc.creator” y en este caso, también debe conocer funciones del lenguaje de programación Java como lo son “stream()” y “anyMatch” .

Es por estas razones que para la implementación de la herramienta desarrollada en esta tesina se optó por no continuar el prototipo ya mencionado, sino comenzar el desarrollo de la herramienta con un nuevo enfoque el cual se explica en los capítulos siguientes. El primer paso fue definir las características y los requerimientos funcionales que debían cumplir la herramienta y el lenguaje que provea la misma, los cuales son explicados en el próximo capítulo.

Capítulo 5 | Especificación de la herramienta

Introducción

Luego de investigar y estudiar el marco teórico explicado en los capítulos precedentes se continuó por analizar las distintas características que debía cumplir la herramienta para poder satisfacer los objetivos planteados en el capítulo 1. En este capítulo se comienza por explicar las características que debe cumplir la herramienta, las cuales tienen como objetivo facilitar su uso y aprendizaje incluso para aquellos usuarios sin conocimientos en programación, continuando por los requerimientos funcionales que debe proveer, los cuales le permitirán satisfacer los objetivos planteados.

Características

Como fue expresado en el capítulo 1, el objetivo principal de esta tesina es el de crear un DSL (Voelter, M. 2013) que pueda ser usado por usuarios sin conocimientos informáticos, para ello el lenguaje creado debe tener ciertas características detalladas a continuación:

- Sencillo: Al ser pensado para usuarios no informáticos el lenguaje no debe requerir conocimientos relacionados a ningún lenguaje de programación, en el caso de DSpace el lenguaje en cuestión es Java, esto significa que para la utilización del lenguaje y la herramienta el usuario no debe conocer nombres de métodos ni tipos de variables, entre otros.
- Potente: El lenguaje debe tener un equilibrio entre la simplicidad y la potencia, dado que si se hace demasiado énfasis en la facilidad de uso y muy poco en la potencia la herramienta no será de gran utilidad, dado que las acciones que se puedan realizar

serán pocas y simples. Por otro lado es importante no descuidar la simplicidad, dado que si la herramienta permite realizar variadas y potentes acciones pero con una sintaxis compleja, los usuarios no informáticos encontrarán el aprendizaje y el uso de la herramienta difícil y agobiante, por lo que podrían desistir de su uso.

- Abstracto: El lenguaje debe poder utilizar funciones Java pero lograr abstraer al usuario de ellas, esto tiene como objetivo obtener la potencia y facilidades que brinda el lenguaje de programación Java pero a su vez simplificar la sintaxis de la herramienta para que el usuario se abstraiga de lo que ocurre por detrás y no necesite conocimientos de programación para poder utilizarla.
- Expresivo: El lenguaje debe ser lo suficientemente expresivo para que al mirar una consulta se pueda, por lo menos, tener un concepto de lo que esa consulta va a hacer. Esta característica debe tener un equilibrio con la sencillez y simplicidad del lenguaje, esto se debe a que al querer construir un lenguaje muy expresivo se puede perder estas otras características. Por ejemplo si una consulta se torna muy larga y con muchas palabras claves, dicha consulta será muy expresiva pero su simplicidad en la escritura se verá afectada.

Requerimientos funcionales

La herramienta desarrollada en esta tesina debe proveer un lenguaje de consulta que cumpla con los objetivos planteados (ver capítulo 1). Para lograrlo se analizaron los casos de uso y se dividieron en tareas más simples a fin de determinar acciones concretas que el lenguaje debía poder ejecutar. De esta manera se detallan los requerimientos funcionales del lenguaje desarrollados en esta tesina.

Validación

El lenguaje debe permitir ejecutar distintas validaciones sobre los elementos del sistema (Comunidad, Colección e Ítem). Una validación puede definirse como la verificación de que un elemento posee un valor que se encuentra dentro de un dominio específico. Expresado en otras palabras, la validación puede definirse como la verificación de una o varias condiciones sobre un elemento, es decir verificar si un elemento en específico cumple con ciertas condiciones o reglas. Llevando esta definición a un caso concreto en los repositorios digitales sobre los cuales estaba basada esta tesina, una validación puede definirse como la verificación de que un elemento del dominio ya sea un Item, una Colección o una Comunidad cumplan con ciertas condiciones o reglas.

En base a esta definición y a los casos de uso explicados anteriormente se plantean las validaciones que permite la herramienta desarrollada en esta tesina. Dichas validaciones están orientadas a analizar los metadatos que poseen los elementos del dominio de DSpace, esto llevó a dividir las las validaciones en dos categorías, validaciones por valor de metadato y validaciones por existencia de metadato.

Validaciones por valor de metadato

Las validaciones permiten al usuario verificar si el valor de un metadato cumple o no con ciertas condiciones, para permitir esto se definieron distintas condiciones básicas:

- Igualdad: Esta condición permite verificar si un metadato tiene valor igual a otro valor
- Desigualdad: Esta condición permite verificar si un metadato tiene valor distinto a otro valor

- Parecido (Like): Esta condición permite verificar si el valor de un metadato contiene a otro valor
- No parecido (not Like): Esta condición permite verificar si el valor de un metadato no contiene a otro valor
- Mayor: Esta condición permite verificar si el valor de un metadato es mayor a otro valor
- Menor: Esta condición permite verificar si el valor de un metadato es menor a otro valor

Existencia de metadato

Este tipo de condición permite verificar si un elemento del sistema posee un metadato específico, a su vez es necesaria la funcionalidad opuestas, es decir aquella que valide si un elemento no contiene un metadato, esta funcionalidad es útil para verificar si existe algún elemento del sistema que no contenga los metadatos obligatorios.

Selección

La herramienta desarrollada debe proveer un mecanismo de selección que permita a los usuarios recuperar distintos elementos que se encuentren en el repositorio, la selección de los elementos está basada en las distintas condiciones mencionadas en la sección anterior lo que le permite a los usuarios que utilicen la herramienta recuperar elementos ya sea para realizar búsquedas, como por ejemplo recuperar elementos por autor o recuperar elementos que no cumplan con ciertos requisitos, por ejemplo seleccionar los Ítems que no tengan título. Para permitir esto, el módulo de consultas de selección que provee el lenguaje desarrollado en esta tesina permite seleccionar elementos en base a 2 tipos de condiciones:

- Condiciones de validación
- Condiciones por handle

La primera condición mencionada le permitiría al usuario agregar tantas consultas de validaciones como sea necesario, las cuales fueron explicadas en la sección anterior, esto brinda la posibilidad de validar distintos metadatos en simultáneo. Además, el módulo de selección agrega la segunda condición mencionada, la cual le permitiría al usuario seleccionar elementos utilizando el identificador único handle. A continuación se explicaran ambas condiciones con más detalle.

Selecciòn por validaciones

El módulo de selección por validaciones permite a los usuarios recuperar distintos elementos que cumplan con ciertas características o condiciones, a través de esta funcionalidad los usuarios pueden realizar búsquedas, identificar elementos mal cargados o que posean información incorrecta. Para realizar las consultas mencionadas los usuarios pueden agregar las validaciones explicadas en la sección anterior, con el objetivo de que los administradores puedan ser lo mas especifico posible al momento de escribir la consulta y seleccionar los elementos deseados, el módulo permite que se agreguen tantas reglas de validación como sea necesario, lo que brinda una gran potencia, precisión y una amplia gama de consultas. Por ejemplo es posible seleccionar Ítems que no contengan el metadato "*dc.title*" y el autor del Ítem sea "*Marisa De Giusti*". Este caso concreto permite verificar si existe algún elemento cargado en el repositorio cuyo autor sea Marisa De Giusti y no posea título, lo cual indicaría que hubo un error en la carga del mismo y es importante arreglarlo.

Selecciòn por handle

Esta funcionalidad realiza dos aportes importantes al módulo de selección. Por un lado permite recuperar elementos por su handle, es decir permite seleccionar Ítems, Colecciones o Comunidades por su identificador persistente. Por ejemplo seleccionar el Ítem con handle 11746/5098.

Antes de explicar la siguiente funcionalidad que brinda la selección por handle es necesario hacer una referencia al capítulo 3 subsección “Modelo”, en donde se explica el modelo del software DSpace el cual contiene Comunidades que pueden contener otras Comunidades y Colecciones y a su vez las colecciones contienen Ítems. En este contexto la selección por handle permite al usuario recuperar elementos en base al handle del elemento que los contiene, por ejemplo se podrìan recuperar todos los Ítems que se encuentren en la Colección o Comunidad con el handle especificado. Para dar un ejemplo concreto en el repositorio digital CIC-Digital (Digital.cic.gba.gob.ar. 2017) existe una Comunidad llamada “Centros” cuyo handle es 11746/3, por lo tanto un usuario puede seleccionar todos los Ítems que se encuentren dentro la comunidad con handle 11746/3, en este caso la herramienta deberá recuperar todos los Ítems que se encuentren en las subcomunidades y colecciones de dicha comunidad.

Selección por handle y condiciones

Con el objetivo de generar consultas con mayor poder discriminatorio–y de este modo brindar al usuario un mayor precision y control sobre los elementos que van a ser recuperados, el módulo de selección permite combinar los dos métodos explicados anteriormente, es decir que en una misma consulta se puede seleccionar por handle y a su vez agregar distintas validaciones. Por ejemplo un usuario puede seleccionar los Ítems que se encuentran en la Comunidad con handle 11746/3, que en el repositorio CIC-Digital se corresponde con la Comunidad Centros, y que a su vez tengan de autor a Marisa De Giusti y que en el metadato dcterms.abstract contengan (like) el valor “preservación”.

Transformación

Finalmente la herramienta desarrollada debe proveer un mecanismo que permita transformar todos los elementos previamente seleccionados con el fin de corregir valores erróneos en los metadatos, agregar metadatos faltantes o eliminar metadatos que no corresponden. Además como esta herramienta está pensada para que los distintos administradores del sistema puedan escribir nuevas consultas sin la necesidad de acudir a un programador, existe la posibilidad de que la consulta escrita tenga efectos secundarios que son ignorados por el usuario al momento de escribirla, es por esta razón que la herramienta también debe proveer un mecanismo pensado para minimizar este tipo de errores, para cumplir este objetivo la herramienta debe mostrar una vista anticipada de los elementos que van a ser modificados junto con el valor actual del metadato a modificar y el nuevo valor del metadato, lo cual le brinda al usuario la posibilidad de corregir la consulta antes de ejecutarla hasta que el resultado sea el deseado.

Modificar, agregar y eliminar metadatos

Como la carga de los elementos del repositorio es realizada manualmente por los administradores, existe la posibilidad de que se cometa un error y algún metadato quede con un valor erróneo, que no sea cargado o que se agregue alguno que no corresponde. A

lo largo del tiempo estas situaciones pueden darse en varios elementos simultáneamente, por ejemplo Ítems y si bien es posible recorrer cada Ítem y modificarlo individualmente esta tarea es tediosa y requiere de mucho tiempo, es por esta razón que la herramienta permite realizar modificaciones en el valor de uno o más metadatos, asimismo también permite agregar y eliminar uno o varios metadatos, lo que permite corregir los errores mencionados de manera rápida y simple.

Referencia a otro metadato

Como ya fue mencionado, es posible que se produzcan errores al momento de cargar nuevos elementos en el repositorio, uno de los casos posibles es la carga de contenido en el metadato incorrecto, para dar un ejemplo concreto que se puede dar en CIC-Digital, es posible cargar el nombre del autor de la obra (metadato dcterms.creator.author) en el metadato que indica el nombre del director (dcterms.contributor.director), en este caso la solución es copiar el valor del metadato dcterms.contributor.director en el metadato dcterms.creator.author. Para resolver este problema la herramienta permite, en las acciones de agregar y modificar, referenciar el valor de otro metadato. Es decir que al momento de escribir la consulta que agrega un nuevo metadato es posible asignarle como valor el valor de otro metadato ya existente.

Expresiones regulares

Al momento de realizar una modificación sobre un metadato es posible querer modificar el contenido completo o querer modificar solo una parte de este, dado que la mayoría de los errores son pequeños por ejemplo una palabra o un enlace mal escrito. Para lograr esto último la herramienta debe contar con una subselección, es decir poder seleccionar solo una parte del contenido del metadato. Una forma de lograrlo es a través de expresiones regulares, si bien su uso requiere un cierto pero no elevado conocimiento en programación, su utilización permite resolver el problema planteado así como también que las subselecciones puedan ser mucho más específicas y finas.

Vista anticipada

Es posible que al momento de escribir una consulta de modificación, ya sea escrita por un administrador o un programador, no se tenga total conciencia de todos los elementos que serán modificados, esto puede llevar a modificaciones no deseadas, a inconsistencias en la base de datos y a tener que generar nuevas consultas que arreglen los errores de las anteriores, entre otros. Con el objetivo de evitar que esto suceda la herramienta debe presentarle al usuario una vista anticipada de la modificación a realizar, dicha vista debe mostrar principalmente el handle del elemento a modificar, el nombre del metadato a modificar, el valor actual del mismo y el nuevo valor. Esta funcionalidad, aunque no es suficiente (ver trabajos futuros), sirve como una primera medida para evitar errores involuntarios, dado que el usuario puede revisar todos los elementos que van a ser modificados, sus valores actuales y como quedaran los valores luego de ejecutar la consulta, en este punto el usuario debe poder elegir si desea ejecutar la consulta o modificarla, lo que debe llevar a una nueva vista anticipada.

Retroceso automático

Algunas de las operaciones que se pueden ejecutar con la herramienta desarrollada en esta tesina pueden involucrar grandes cantidad de elementos, por ejemplo se puede realizar una consulta de modificación sobre 200 Ítems, en estos casos es posible que la consulta falle sin haber modificado a todos los elementos involucrados en la misma, si esto ocurre y no se toman las medidas correspondientes se deberían revisar los Ítems uno por uno para ver el estado en el que quedaron, ya que, al haber fallado la consulta, no puede asegurarse que todos los Ítems fueron modificados, así como tampoco puede asegurarse que aquellos ítems que sí fueron modificados quedaron con el valor deseado. Para evitar esto, la herramienta debe, en caso de un fallo en medio de la ejecución de una consulta, realizar un retroceso automático al estado previo a la ejecución de la consulta sobre todos los elementos modificados. Esto evita los estados inconsistentes, las modificaciones erróneas y la tarea de revisar el estado de cada elemento, permitiendo al administrador enfocarse solamente en la identificación y corrección del error.

Una vez definidas las características y los requerimientos funcionales que la herramienta debía satisfacer, se continuó por la creación de diversas sintaxis para el lenguaje de consulta y posteriormente la elección de una de ellas, para efectuar dicha decisión fueron tomadas en cuenta tanto la simplicidad como la expresividad de cada una de las posibles sintaxis. Con la sintaxis elegida se continuó por el desarrollo del diagrama UML (Lenguaje Unificado de Modelado) junto con la posterior implementación del lenguaje propiamente dicho. Este procedimiento es explicado en el capítulo siguiente, el cual detalla entre otras cosas las distintas sintaxis planteadas, la elección y justificación de la sintaxis elegida, el UML final de la herramienta, los patrones de diseño aplicados para el desarrollo del mismo junto con el flujo de información.

Capítulo 6 | Implementación del lenguaje de consulta

Introducción

Con el análisis terminado sobre las características y requerimientos funcionales que una herramienta pensada para satisfacer los objetivos planteados en el capítulo 1 debía cumplir y atentos al desarrollo de una herramienta que cumpliese con ellos, se comenzó con la implementación de la misma. En este capítulo se explican las distintas etapas de desarrollo comenzando por cómo se utilizó de la JSR-341 y terminando con la sintaxis final de la herramienta junto con ejemplos de las distintas operaciones que permite.

Utilización de la JSR-341

En el capítulo 4 de esta tesina se explicaron las distintas ventajas y características que provee la JSR-341, siendo una de ellas un lenguaje de expresiones que permite, entre otras cosas, acceder a los objetos Java desde la capa de presentación. Asimismo se explicaron los problemas que presenta el prototipo de herramienta desarrollado en (Terruzi, 2015) el cual utiliza el lenguaje mencionado como lenguaje de consulta para acceder directamente a los objetos Java. Es por lo analizado en dicho capítulo que se decidió por crear una sintaxis de lenguaje de consulta propio y utilizar la JSR-341 de una manera diferente.

Una de las tantas funcionalidades que provee la JSR-341 es la de acceder a funciones Java implementadas como métodos estáticos, para hacer esto solo es necesario indicarle a la JSR-341 un nombre clave por cada función a referenciar, dicho nombre debe tener la siguiente estructura:

"ejemplo:ejemplo"

Como puede verse el nombre clave que se le asigna a la función consta de dos sub palabras claves separadas por el carácter ':'. Es decir si el objeto Java Ítem tuviese una función estática llamada "agregarMetadato" se puede configurar la JSR-341 para que referencie a esta función con la palabra clave "item:agregarMetadato", de esta manera los administradores solo tendrían que indicarle a la JSR-341 el nombre clave de la función que quieren ejecutar, esto permite restringir las funciones de los objetos Java que pueden ser ejecutadas por los usuarios finales, impidiendo que ejecuten funciones potencialmente peligrosas, como por ejemplo eliminar elementos del repositorio. Esta simple configuración se realiza a través de una clase de la JSR-341 llamada "ELProcessor" la cual provee la API[1] necesaria para poder utilizar todas las funcionalidades de la JSR-341, para utilizarla solo se debe crear una instancia de la misma. Es a través del método "defineFunction" de dicha clase que se define la relación entre palabra clave y método al cual corresponde esa palabra. Puede verse un claro ejemplo a continuación:

processor.defineFunction("seleccionar","item","org.dspace.app.xmlui.aspect.ELProcessor.SelectionAction", "selectItems");

En el ejemplo dado se puede apreciar que el método "defineFunction" recibe 4 parámetros, como se explicó anteriormente el nombre clave consta de dos palabras separadas por el símbolo ":", el primer parámetro que recibe la función corresponde con la primera palabra del nombre clave (es decir con la palabra que se ubica antes del carácter ":"), mientras que el segundo parámetro corresponde con la segunda palabra del nombre clave (es decir con la palabra que se ubica después del carácter ":"), el tercer parámetro se corresponde con el nombre de la clase a la cual queremos hacer referencia, mientras que el cuarto parámetro es el nombre del método de dicha clase. Para clarificar el ejemplo dado, la palabra clave creada sería "seleccionar:item" la cual ejecutaría el método "selectItems" de la clase "SelectionAction".De esta manera es como se pueden crear diferentes referencias a múltiples métodos de una clase o a múltiples métodos de distintas clases. Sin embargo esta implementación presenta 2 problemas, en primer lugar, como se dijo anteriormente, para que la JSR-341 pueda referenciar una funcion Java a través de un nombre clave esta tiene que ser estática y esta condición no la cumplen todas funciones de DSpace (este problema será resuelto más adelante en esta tesina en el capítulo 7 sección "Actions"), el segundo problema que se presenta es la creación de una sintaxis simple y clara para el administrador, es decir que las palabras claves por las cuales el usuario pueda acceder a los métodos sean simples de recordar, intuitivos y fáciles de escribir. Este último problema será tratado en la sección siguiente.

[1] Del inglés: Application Programming Interface, es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Creación y elección de la sintaxis

Como se dijo en la sección anterior, se optó por utilizar la funcionalidad que provee la JSR-341 que permite referenciar funciones estáticas Java para el desarrollo de la sintaxis del lenguaje de consultas que provee la herramienta implementada en esta tesina. El objetivo de utilizar esta funcionalidad es abstraer al usuario final, es decir a los administradores del repositorio, del funcionamiento interno de la herramienta y de la implementación y métodos que poseen los objetos del dominio de DSpace. Al decidir utilizar la funcionalidad para crear la sintaxis del lenguaje de consulta se presentan distintas características que debe cumplir dicha sintaxis:

- Debe ser simple de escribir
- Debe ser expresivo
- No debe requerir grandes conocimientos en programación para poder ser utilizada

Para poder diseñar una sintaxis que cumpliera con las condiciones mencionadas se optó por crear dos prototipos y escribir consultas de selección y de transformación con cada uno de ellos, para finalmente compararlos en base a las condiciones establecidas y elegir aquel que se adapte mejor a ellas; las consultas de validación se abordarán más adelante en este mismo capítulo.

Como primera opción se optó por una sintaxis que sugiere una concatenación de condiciones, como primer ejemplo se presenta una consulta que permita seleccionar Ítems en base a 2 condiciones:

seleccionar:condicion(condicion1).condicion(condicion2).item()

Para analizar la consulta se la dividirá en 3 partes, la primera parte abarca todo lo anterior al carácter ‘:’ en este caso es la palabra clave “seleccionar”, esta palabra indica el tipo de consulta a realizar. Luego se pueden observar las condiciones por las cuales los elementos serán evaluados, como puede verse en el ejemplo para agregar varias condiciones solo hace falta concatenar una condición luego de otra, finalmente se indica el tipo de DSO que será seleccionado. Es decir que si se quisiera realizar la misma consulta pero seleccionar Colecciones en lugar de Ítems y agregar una condición más la consulta sería la siguiente::

seleccionar:condicion(condicion1).condicion(condicion2).condicion(condicion3).coleccion()

Para el ejemplo de una consulta de transformación, siguiendo la misma línea sintáctica, se muestra una consulta que transformaría los Ítems previamente seleccionados

seleccionar:condicion(condicion1).condicion(condicion2).transformar()
.modificacion(modificacion1).item()

Se puede apreciar a simple vista que la consulta de transformación es una concatenación a la consulta de selección. Para transformar los elementos previamente seleccionados se debe agregar la palabra clave “transformar” y luego concatenar todas las transformaciones o modificaciones que se quieran realizar y finalmente, al igual que en la consulta de selección, indicar el tipo de DSO que se quiere modificar.

Como segunda opción para la sintaxis del lenguaje de consultas se optó por una sintaxis que sugiere un listado de condiciones, como primer ejemplo se presenta una consulta de selección que pemitiría seleccionar Ítems en base a 2 condiciones:

seleccionar:item(condicion1, condicion2)

Al igual que en la opción anterior esta consulta también puede ser dividida en 3 partes, la primera representa la acción que se desea realizar y se ubica antes del carácter ':', en este caso es la palabra clave "seleccionar". Luego de los dos puntos se indica el tipo de DSO a seleccionar, en este caso "item" y finalmente entre paréntesis se indican, separadas por comas, todas las condiciones por las que se desea filtrar. Al igual que la opción anterior, para seleccionar por Colecciones en lugar de por Ítems solo se debe cambiar la palabra "item" por la palabra "coleccion", además si se quisiera agregar una tercera condición solo se debería agregar dentro de los paréntesis y separarla por una coma:

seleccionar:coleccion(condicion1, condicion2, condicion3)

Para el ejemplo de una consulta de transformación, siguiendo la misma línea sintáctica, se muestra una consulta que transformaría los Ítems previamente seleccionados

transformar:item(condicion1, condicion2 - modificacion1, modificacion2)

Se puede apreciar que la consulta de transformación no varía demasiado de la consulta de selección. Aún es posible separar la consulta en 3 partes, la primera que indica el tipo de consulta a ejecutar, en este caso como es una consulta de transformación la palabra clave es "transformar"; la segunda parte de la consulta indica el elemento que queremos transformar, en este caso es Ítem y finalmente la tercera parte presenta el único cambio con respecto a la consulta de selección, podemos ver que dentro de los paréntesis hay condiciones y modificaciones las cuales se encuentran separadas por un guión del medio '-'. Dicho carácter indica la separación entre las condiciones de selección y las modificaciones, las condiciones por las que el usuario quiera filtrar se encontrarán antes del guión medio, mientras que las modificaciones que el usuario quiera realizar sobre los elementos seleccionados se encontrarán después del mencionado carácter. Como puede apreciarse en el ejemplo dada una misma consulta de transformación puede tener una o más condiciones de selección y puede realizar una o más modificaciones en simultáneo.

Una vez planteadas las distintas opciones de sintaxis para el lenguaje de consulta y luego de escribir prototipos de consultas de selección y transformación con cada una de ellas, se prosiguió por compararlas y elegir la sintaxis que se adecuara más a las características planteadas al comienzo de esta sección. En primer lugar se comparan las consultas de selección:

Opción 1: *seleccionar:condicion(condicion1).condicion(condicion2).item()*

Opción 2: seleccionar:item(condicion1, condicion2)

La opción 2 al tener todas las condiciones dentro de los paréntesis y separarlas solo por el carácter coma ‘,’ resulta más corta y simple al momento de escribirla y más clara al momento de analizarla, mientras que la primera opción al necesitar separar cada una de las condiciones y escribir la palabra clave “condición” hace que la consulta de selección quede más larga y menos legible al momento de analizarla. Por otro lado la concatenación de condiciones que presenta la primera opción es algo muy visto y conocido en el área de la programación pero puede que no sea tan intuitivo para un administrador sin conocimientos en dicha área.

Antes de optar por una opción para el lenguaje de consulta se analizarán los prototipos de consultas de transformación:

Opción 1: *seleccionar:condicion(condicion1).condicion(condicion2).transformar().modificacion(modificacion1).item()*

Opción 2: *transformar:item(condicion1, condicion2 - modificacion1)*

Se puede notar que la desventaja de la concatenación en la opción 1 es ahora incluso más grande, mientras más se agranda la consulta, esta se volverá más larga e ilegible. Por otro lado la opción 2 mantiene las ventajas de la legibilidad y simplicidad explicadas anteriormente, sin embargo presenta un inconveniente, el cual es la necesidad de que el usuario final entienda, identifique y pueda aplicar la división que existe dentro del paréntesis entre las condiciones para la validación y las modificaciones que se quieran realizar. Esto último se debe, entre otras cuestiones, a que el carácter guión medio ‘-’ no sea lo suficientemente expresivo o intuitivo, igualmente este problema será tratado más adelante en el capítulo Trabajos futuros de esta tesina.

Finalmente, luego de comparar ambas opciones y cada una de sus prototipos de consultas se optó por la elección de la segunda opción, dado que resulta más simple y corta su escritura y no posee concatenación que, como se dijo anteriormente, puede que sea intuitivo para los programadores pero puede no serlo para usuarios sin conocimientos en el ambiente de programación como lo son los administradores de los repositorios. A su vez, con el objetivo de hacer que la sintaxis sea más flexible y simplificar su escritura, se optó por no poner restricciones en cuanto al espacio que existe entre los paréntesis y las condiciones ni entre los espacios que se deben dejar entre las condiciones y las comas que las separan. Para clarificar lo dicho se presenta el siguiente ejemplo en el cual se tiene dos consultas equivalentes en cuanto a funcionalidad pero levemente diferentes sintácticamente:

seleccionar:item(condicion1 , condicion2)

seleccionar:item(condicion1,condicion2)

Como puede verse las únicas diferencias que existen entre las dos consultas planteadas son los espacios en blanco entre los paréntesis y las condiciones, y los espacios en blanco entre las condiciones y la coma que las separa. El hecho de no tener restricciones en cuanto a estos espacios en blanco simplifican la escritura de la consulta dado que el administrador no va a tener la necesidad de preocuparse por ellos, es decir que no va a tener que revisar la consulta en busca de este tipo de errores sintácticos, lo cual es

algo muy comùn en el ambiente de programaciòn. Con esto se busca tambièn utilizar la experiencia que se tiene como programador para tratar de evitarle a los administradores problemas que se sabe son tediosos y difíciles de detectar con el objetivo de abstraer a los administradores todavía más del ambiente de la programación.

Creación de las consultas

Como se dijo en el capítulo 5 subsección “requerimientos funcionales” , la herramienta desarrollada en esta tesina debe proveer un lenguaje de consulta que permita realizar validaciones, selecciones y transformaciones sobre los elementos del repositorio. Para ello en el capítulo mencionado se plantearon los requerimientos funcionales y las características que debía cumplir dicho lenguaje, el paso siguiente fue la creación y elección de la sintaxis a utilizar. Una vez planteados los requerimientos y características, y definida la sintaxis se prosiguió por la creación de las distintas consultas que el lenguaje iba a permitir.

Consulta de validación

Como fue expresado en los capítulos precedentes, las consultas de validaciones le permitirán al usuario validar distintas características sobre los elementos del sistema a fin de verificar, por ejemplo, que los elementos posean todos los metadatos obligatorios. En el capítulo 5 de esta tesina se listaron los distintos requerimientos funcionales que debe cumplir la herramienta, a continuación se volverán a listar solo los requerimientos funcionales de las consultas de validación y a su vez se mostrará la consulta correspondiente a cada requerimiento.

- Igualdad
- Desigualdad
- Parecido (Like)
- No parecido (not Like)
- Mayor
- Menor
- Existencia de metadato

Igualdad

Esta condición sirve para comparar el valor de un metadato con otro valor, la validación será correcta si ambos valores son exactamente iguales. Por lo tanto para poder escribir esta condición el usuario debería indicar por lo menos el nombre del metadato, el valor con el cual lo quiere comparar y un identificador que le diga a la herramienta que la condición que quiere evaluar es la igualdad. Se presenta a continuación el formato de la consulta que debe escribir el usuario para realizar una validación de igualdad:

metadato=nuevo valor

Al desarrollar el formato de la consulta se buscó que sea simple e intuitivo incluso para los administradores sin conocimientos en programación, siguiendo este formato si se

quisiera validar que el metadato “dc.title”, el cual indica el título de un elemento, tenga el valor “un título de ejemplo” la consulta a escribir sería la siguiente:

dc.title= un título de ejemplo

Desigualdad

Esta condición es la opuesta a la igualdad, en otras palabras sirve para comparar el valor de un metadato con otro valor pero, a diferencia de la igualdad, será correcta si ambos valores son distintos. Al igual que en la validación de igualdad para poder escribir esta condición el usuario debería indicar por lo menos el nombre del metadato, el valor con el cual lo quiere comparar y un identificador que le diga a la herramienta que la condición que quiere evaluar es la desigualdad. Se presenta a continuación el formato de la consulta que debe escribir el usuario para realizar una validación de desigualdad:

metadato^=nuevo valor

Cabe destacar que originalmente el símbolo que iba a utilizarse para identificar a la consulta de desigualdad iba a ser “!=” pero la JSR-341 tiene reservado el carácter “!” por lo que este tuvo que ser reemplazado y se optó por utilizar el carácter “^”, quedando como símbolo que identifica a la consulta de desigualdad “^=”. Por ejemplo si se quisiera validar que el metadato “dc.title” no tenga el valor “un título de ejemplo” la consulta a escribir sería la siguiente:

dc.title^= un título de ejemplo

Parecido (Like)

Esta condición permite identificar si el valor de un metadato contiene a otro valor, es decir que la evaluación de la condición será correcta si el valor del metadato indicado contiene al menos una vez a el otro valor. Al igual que las validaciones antes mencionadas para escribir esta validación también son necesarios 3 elementos, el metadato, el valor que se quiere evaluar y un identificador que le indique a la herramienta que se quiere realizar una consulta de parecido (like). El formato de esta validación es el siguiente:

metadato~valor

Como puede verse el formato de esta consulta es similar a los anteriores, cambiando solamente en el identificador de la validación. Esto tiene como objetivo que los usuarios no sientan que todas las consultas son totalmente distintas y facilitarles recordarlas y escribirlas. Si se quisiera verificar si el metadato “dc.title” contiene el valor “título” la consulta a escribir es la siguiente:

dc.title~título

No Parecido (Not Like)

Esta validación es la opuesta al parecido (like), permite verificar si el contenido del metadato indicado no contiene un valor determinado, es decir que esta validación será

correcta cuando el contenido del metadato no contenga el otro valor. La estructura de esta validación también consta de el metadato a evaluar, el valor por el cual se quiere evaluar y el identificador de la evaluación. Se presenta la estructura de esta validación:

metadato^~valor

Como puede verse para la creación del identificador de esta validación se utilizó el mismo símbolo que identifica la consulta Parecido (Like) "~", y al igual que en la validación de Desigualdad se le agregó el carácter "^" para indicar la negación. Si se quisiera verificar si el metadato dc.title no contiene el valor "título" se debería escribir la siguiente consulta:

dc.title^~título

Mayor

Esta validación permite al usuario verificar si el valor de un metadato es mayor a otro valor, es decir que la validación será correcta cuando el valor del metadato sea mayor al otro valor. Al igual que las validaciones anteriores esta validación también consta de 3 partes, el nombre del metadato, el valor que se quiere verificar y el identificador que le sirve a la herramienta para saber que se quiere aplicar la validación de Mayor. La estructura de esta validación es la siguiente:

metadato>valor

Se eligió el carácter '>' como identificador de esta validación dado que se espera que resulte intuitivo incluso para aquellos usuarios sin conocimientos en programación. Si se quisiera validar si el metadato dcterms.extent, el cual contiene la cantidad de páginas que posee el archivo de un Ítem, es mayor a 10 se debería escribir la siguiente consulta:

dcterms.extent>10

Menor

Esta validación permite al usuario verificar si el valor de un metadato es menor a otro valor, es decir que la validación será correcta cuando el valor del metadato sea menor al otro valor. Al igual que las validaciones anteriores esta validación también consta de 3 partes, el nombre del metadato, el valor que se quiere verificar y el identificador que le sirve a la herramienta para saber que se quiere aplicar la validación de Menor. La estructura de esta validación es la siguiente:

metadato<valor

Al igual que en la validación anterior se eligió este carácter porque se espera que sea intuitivo incluso para aquellos usuarios sin conocimientos en programación, además dado que en la validación de Mayor se utilizó el carácter '>' resulta lógico utilizar el carácter '<' para la validación de Menor. Utilizando el mismo ejemplo que en la validación de Mayor la consulta sería la siguiente:

dcterms.extent<10

Existencia de metadato

Esta validaciòn permite al usuario verificar si el elemento posee el metadato indicado, es decir que esta validación sera correcta cuando el elemento posea el metadato indicado, sin importar el valor del mismo. A diferencia de las validaciones previas esta validación solo consta de 1 parte, la cual es el nombre del metadato cuya existencia se quiere verificar. Por ejemplo si un administrador quisiera verificar si un elemento posee el metadato “dc.title” la validaciòn a escribir serìa la siguiente:

dc.title

Inexistencia de metadato

Esta validación es la opuesta a la validación anterior, permite seleccionar aquellos elementos que no posean el metadato especificado. La estructura de la consulta es igual a la de Existencia de metadato, la diferencia que existe entre ambas es que a esta validación se le agrega el carácter de negación, es decir ‘^’. Como ejemplo se presenta la consulta que valida que el elemento no contenga el metadato dc.title

^dc.title

Durante la creaciòn de las distintas condiciones de validación uno de los grandes objetivos a tener en cuenta fue mantener la simplicidad para que los administradores, sin importar sus conocimientos en programación, pudieran utilizar la herramienta y que utilizarla les resulte lo más intuitivo y fácil posible. Con este objetivo en mente se optó por, al igual que en las consultas de selección, no poner restricciones en cuanto a los espacios en blanco, es decir que en cualquiera de las validaciones mencionadas no existen restricciones en cuanto a los espacios en blanco entre el metadato, el identificador de la consulta y el valor por el cual se quiere evaluar. Para clarificar lo expresado se presenta un ejemplo con dos consulta iguales en cuanto a funcionalidad pero ligeramente diferentes en cuanto a sintaxis:

dc.title = mi título
dc.title=mi título

Ambas consultas realizan la misma validación, verificar que el metadato “dc.title” sea igual al valor “mi título”, esta medida brinda facilidad de escritura y flexibilidad a las consultas . Para complementar las medidas explicadas también se optó por mantener consistencia al momento de indicar la negación de una validación, es decir utilizar el mismo carácter en todas los tipos de validaciones para indicar la negación, lo que brinda facilidad de aprendizaje y de uso a la herramienta. Como ya fue expresado en la secciòn Creación y elección de la sintaxis de este mismo capítulo evitar restricciones en los espacios en blanco simplifica la escritura de la consulta y permite abstraer en cierta medida al administrador del ambiente de la programación. Asimismo también se intentó mantener la misma estructura con el objetivo de simplificar el uso y aprendizaje de la herramienta, siendo que no es lo mismo tener que aprender una sola estructura a tener que aprender una estructura nueva

por cada condición de validaciòn que se quiera realizar. A continuación se detalla la creación de las consultas de selección.

Consulta de selecciòn

¿Cómo se puede ejecutar una consulta de validación si no se puede indicar qué elementos se quieren validar? Es aquí donde aparecen las consultas de selección, como se dijo en capítulos precedentes las mismas sirven para recuperar un grupo de elementos del sistema. Es combinando las consultas de selección con las de validación que se pueden cumplir los requerimientos funcionales planteados en el capìtulo anterior subsección Selección. Los requerimientos mencionados son:

- Selección por validaciones
- Selección por handle
- Selección por handle y validaciones

Selección por validaciones

Estas consultas permiten al usuario seleccionar elementos del sistema que cumplan con las validaciones/condiciones indicadas. Para realizar esto se combinan las consultas de selección junto con las consultas de validación:

Consulta de selección: *seleccionar:item(condicion)*

Consulta de validación: *metadato identificador valor*

Por ejemplo si se quisiera recuperar todos los Ìtems cuyo dc.title es igual a “mi tìtulo” la consulta a escribir serìa:

seleccionar:item(dc.title=mi tìtulo)

Como puede verse para la realización de esta consulta se combinó la consulta de selección junto con la consulta de validación por igualdad explicada en la sección anterior. Esto puede aplicarse con cualquier consulta de validación e incluso se puede utilizar múltiples consultas de validación en una misma consulta de selección. Por ejemplo si se quisiera recuperar los Ítems cuyo autor es Marisa De Giusti y cuyo título contenga la palabra “preservación” la consulta a escribir serìa:

seleccionar:item(dcterms.creator.author = Marisa De Giusti , dc.title~preservación)

Selección por handle

Como se dijo en el capìtulo 5 en la secciòn “Requerimientos funcionales” subsecciòn “Selecciòn por handle”, esta funcionalidad debe permitir recuperar los elementos del repositorio en base a su handle así como también permitir recuperar elementos en base al handle del elemento que los contiene. Esto debería permitir por ejemplo recuperar todas las Colecciones o Ítems que se encuentren dentro de una Comunidad o bien recuperar cualquiera de los 3 elementos mencionados por su handle. Sin embargo al momento de

realizar su implementación se encontró con una incapacidad que presenta el propio DSpace, DSpace provee mecanismos para poder recuperar cualquier DSO en base a su handle, por ejemplo recuperar una Comunidad por su handle, y a su vez provee un mecanismo para recuperar Ítems en base al handle de una Colección o Comunidad padre, sin embargo no provee un mecanismo para poder recuperar Colecciones en base al handle de su Comunidad padre ni recupera subComunidades en base al handle de sus Comunidades padres, es por esta razón que la funcionalidad que pretende brindar la herramienta desarrollada en esta tesina se ve restringida por el mismo software DSpace, de esta manera la funcionalidad que provee esta herramienta permite seleccionar cualquier DSO por su handle y a su vez permite seleccionar Ítems en base al handle de su Colección o Comunidad padre, para realizar dichas acciones se deben escribir las siguientes consultas:

Selección por el handle del elemento: *seleccionar:item(handle = 11746/5127)*

Selección por el handle del elemento contenedor: *seleccionar:item(handle = 11746/5063)*

La primera consulta devolverá el Ítem cuyo handle sea 11746/5127, mientras que la segunda consulta devolverá todos los Ítems que se encuentren en la colección con handle 11746/5063. Se puede notar que ambas consulta son sintácticamente iguales, lo cual brinda una facilidad de uso ya que el administrador no tendrá que recordar distintas sintaxis para cada una de las formas de selección por handle.

Selección por validaciones y handle

Esta funcionalidad brinda una gran potencia a la herramienta, ya que permite combinar los dos módulos de selección explicados anteriormente, esto permite realizar selecciones más finas y le brinda al usuario una amplia gama de consultas a realizar. Como ejemplo de esta funcionalidad se presenta una consulta de selección la cual recupera los Ítems que se encuentren en la colección con handle 11746/5063 y que no tengan el metadato dcterms.abstract y que el metadato dc.title contenga el valor “preservación”:

seleccionar:item(handle= 11746/5063, ^dcterms.abstract, dc.title^~ *preservación*)

Cabe destacar que con el objetivo de darle todavía más flexibilidad a la herramienta y a las consultas que se pueden crear, no se definió ningún orden específico al momento de agregar condiciones de validaciones o de handle, esto permite que la misma consulta pueda ser escrita de distintas maneras y le brinda al usuario mayor soltura al momento de escribirla dado que no deberá acordarse ni memorizar un orden específico de condiciones. Para ejemplificar lo dicho se presenta una consulta que realiza exactamente la misma acción que la consulta anterior pero que sintácticamente es diferente:

seleccionar:item(dc.title ^~*preservación,* handle =11746/5063 , ^dcterms.abstract)

Consulta de transformación

Las consultas de transformación le permitirán al usuario final modificar los elementos previamente seleccionados, como se dijo en capítulos precedentes esta funcionalidad le permitirá modificar valores erróneos, agregar y eliminar metadatos. En esta sección se listan nuevamente los requerimientos funcionales para la consulta de transformación, propuestos en el capítulo 5 de esta tesina, y se da un detalle de cada uno de ellos explicando a su vez la consulta que debería escribir un administrador para poder realizar cada acción.

- Modificar un metadato
- Agregar un metadato
- Eliminar un metadato
- Referenciar a otro metadato
- Utilizar expresiones regulares
- Tener una vista anticipada

Modificar un metadato

Esta funcionalidad le permite a los usuarios finales modificar el valor de un metadato, esta modificación puede afectar a todo el valor del metadato o solo a una parte de él. Esto es posible gracias a que la herramienta también permite hacer una selección sobre qué parte del valor del metadato va a ser modificada, dando más potencia y flexibilidad a las consultas. Dado que la mayoría de las veces los errores en los metadatos son pequeños, es decir que el error puede ser una palabra o incluso un pequeño conjunto de letras, era necesario que la herramienta permitiese modificar solo una parte del contenido del metadato. Como fue explicado en la sección anterior de este mismo capítulo la estructura de las consultas de transformación es la siguiente:

transformar:item(condicion - transformacion)

Las consultas de transformación constan de 3 partes, la primera parte se encuentra antes del símbolo ‘:’, en este caso es la palabra “transformar” que le indica a la herramienta que se realizará una consulta de modificación. Luego se indica el tipo de DSO a modificar, en este caso es “item” pero podría ser “colección” o “comunidad”. Finalmente dentro los paréntesis se encuentran la última sección, en ella se observan las condiciones de selección, explicadas anteriormente y las condiciones de modificación, ambas separadas por el carácter “-”. Las condiciones de modificación tienen el siguiente formato:

metadato ; valor a modificar ; nuevo valor

Las condiciones de modificación también pueden dividirse en 3 partes, la primera parte indica el metadato a modificar. La segunda parte es opcional e indica qué valor del metadato va a ser modificada, si esta parte es omitida se reemplazará el contenido del metadato por el nuevo valor, en caso contrario esta parte puede ser utilizada para referenciar parte del valor de metadato y en ese caso solo esa parte será modificada. Finalmente la tercera parte de la condición es el nuevo valor, es decir el valor por el cual se reemplazará ya sea la totalidad del valor del metadato o solo la parte referenciada. Para dar un ejemplo concreto de esta funcionalidad se mostrará una consulta que recupera un Ítem

en base a su handle y modifica el valor del metadato “dc.title” el cual posee como valor “un título de eejmplo” y se modificará la palabra “eejmplo” por “ejemplo”:

transformar:item(handle=11746/5063 - dc.title;eejmplo;ejemplo)

Como puede verse la consulta se mantiene simple y fácil de escribir, e incluso podría ser todavía más específica, si en lugar de modificar la palabra “eejmplo” se hiciera referencia solo a las letras “eej”, las cuales son el verdadero error. De esta manera la consulta sería la siguiente:

transformar:item(handle=11746/5063 - dc.title;eej;eje)

Al igual que las condiciones de selección en una misma consulta se pueden poner diferentes condiciones de modificación, es decir que se pueden modificar más de un metadato a la vez. Asimismo, al igual que en la condiciones de selección, no existen restricciones en el orden de las condiciones de modificación ni en los espacios en blanco. Es decir que las siguientes consultas, aunque sintácticamente distintas son funcionalmente equivalentes:

transformar:item(handle=11746/5063 - dc.title;eej;eje, dcterms.abstract;modificación completa)

transformar:item(handle=11746/5063 - dcterms.abstract;modificación completa , dc.title ; eej ; eje)

Como se dijo anteriormente ambas consultas realizan la misma acción, en primer lugar recuperan el Ítem con handle 11746/5063, luego modifican en el metadato “dc.title” el valor “eej” por “eje” y a su vez reemplazan el contenido del metadato “dcterms.abstract” por el valor “modificación completa”. De esta forma se demuestra que tanto los espacios en blanco como el orden de las condiciones no afectan al resultado de la consulta.

Finalmente el módulo de modificación tiene una característica más que lo hace todavía más potente, anteriormente se dijo que en las condiciones de modificación se puede indicar específicamente el valor del metadato que va a ser modificado, pero ¿qué ocurre si ese valor se repite pero solo hay que modificar la primera ocurrencia? Este problema es muy común en el ambiente de la programación es por eso que existen métodos en la programación que permiten hacer un “replaceFirst” o “replaceAll”, en base a esta experiencia la herramienta le permite a los usuarios finales reemplazar solo la primera ocurrencia del valor especificado o todas las ocurrencias. Por defecto la herramienta reemplaza todas las ocurrencias del valor especificado, pero en caso de querer reemplazar solo la primer ocurrencia en lugar de escribir la palabra “transformar” se debería escribir la palabra “transformarPrimera”, por ejemplo si se quisiera modificar solo la primera ocurrencia de la las letras “eej” la consulta a escribir sería:

transformarPrimera:item(handle=11746/5063 - dc.title;eej;eje)

Resumiendo, el módulo de modificación permite reemplazar el valor de un metadato, modificar todas las ocurrencias de una frase o palabra, o incluso modificar solo la primera ocurrencia, a su vez también permite agregar múltiples condiciones de modificación en una

misma consulta sin restricciones en cuanto al orden de las mismas ni a los espacios en blanco que hay que dejar entre ellas. Estas funcionalidades y cualidades hacen que el módulo de modificación sea potente, versátil y flexible, y a pesar de ellas las consultas que se pueden escribir no poseen grandes complejidades en cuanto a su escritura o aprendizaje.

Agregar un metadato

Esta funcionalidad le permite al administrador agregar metadatos a un elemento junto con su correspondiente valor. Esta situación puede darse por ejemplo si un metadato no fue cargado o bien si a lo largo del tiempo se debe agregar un nuevo metadato a los Ítems. Las consultas que agregan metadatos a un elemento poseen la siguiente estructura:

agregar:item(condicion - transformacion)

Como puede verse la estructura de la consulta es igual a la consulta de modificación. Existen 2 diferencias entre estas consultas, la primera es la palabra clave “agregar” y la segunda es la estructura de las condiciones de transformación, en las consultas que permiten agregar metadatos las condiciones de transformación tienen la siguiente estructura:

metadato;valor

Esta estructura es más simple que las condiciones de las consultas de modificación, en este caso las condiciones solo tienen 2 partes, el nuevo metadato a agregar y el valor del mismo. Por ejemplo si se quisiera agregar un metadato al un conjunto de Ítems seleccionados la consulta a escribir sería la siguiente:

agregar:item(handle =11746/186 - dcterms.subject.materia; nueva materia)

Esta consulta, si se ejecuta en el repositorio CIC-DIGITAL, recupera todos los Ítems que se encuentren dentro de la colección con handle 11746/186 (en CIC-DIGITAL es la colección “Artículos, Informes y presentaciones en Congresos” de la comunidad “Universidad Nacional de La Plata (UNLP)”) y les agregaría un metadato dcterms.subject.materia con valor nueva materia.

Al igual que la modificación esta funcionalidad también permite agregar múltiples metadatos en una misma consulta y no posee restricciones en cuanto a los espacios en blanco que hay que poner entre las condiciones.

Eliminar un metadato

Esta funcionalidad le permite a los administradores eliminar un metadato, ya sea porque quedó en desuso o porque fue cargado equívocamente. La estructura de esta consulta es similar a las consultas antes presentadas:

eliminar:coleccion(condicion - transformacion)

Como puede verse se intentó que las estructuras de las consultas sean lo más parecidas posibles. En este caso cambia la primera palabra clave, para indicarle a la herramienta que la consulta será de eliminación se debe poner la palabra “eliminar”, la segunda parte de la consulta se mantiene igual, siendo posible ejecutar consultas de eliminación sobre cualquiera de los 3 tipos de DSO mencionados (Ítem, Colección, Comunidad), y es en la tercera parte donde se genera el mayor cambio con respecto a las consultas antes mencionadas, en específico el cambio se genera en las condiciones de transformación las cuales son ahora todavía más simples, dado que solo es necesario que se indique el metadato a eliminar, es decir que la estructura de las condiciones de transformación para las consultas de eliminación es el siguiente:

metadato

Por lo tanto si un administrador quisiera eliminar el metadato “dc.title” de un Ítem en particular debería ejecutar la siguiente consulta:

eliminar:item(handle = 11746/5063 - dc.title)

En las 3 consultas de transformación mencionadas anteriormente se intentó mantener una estructura similar para facilitarle a los administradores el aprendizaje y utilización de la herramienta. A su vez también se tuvo en mente el objetivo de darle potencia y simplicidad a las consultas, ambas características serán abordadas nuevamente en el capítulo Trabajos futuros de esta tesina. Finalmente para brindarle, justamente, más potencia al módulo de transformaciones, se le agregó la posibilidad de referenciar a otros metadatos, utilizar expresiones regulares y generar una vista previa de la modificación, dichas características serán abordadas a continuación.

Referenciar a otro metadato

Esta funcionalidad le permite a los administradores referenciar el valor de un metadato al momento de realizar la creación o modificación de un metadato, es decir que si se está modificando el metadato “A” se puede asignar como nuevo valor el valor del metadato “B”, esta acción podría ser útil por ejemplo se tiene el metadato “dc.subject” y “dc.subject.acmcss98”, se decidió no usar más el segundo dado que no beneficia casi nada a la organización de los documentos. No obstante, no se quiere perder los que ya están catalogados, por ello la solución sería pasar todos los “dc.subject.acmcss98” a un dc.subject y luego eliminar el campo que no se usa más.

Utilizar expresiones regulares

Como se dijo previamente, al momento de modificar un metadato la herramienta le permite a los administradores seleccionar que parte del valor del metadato va a ser modificada, esto le brinda más potencia y flexibilidad a la herramienta y a su vez permite realizar pequeños cambios en el contenido de los metadatos en consultas simples y cortas. Sin embargo esta funcionalidad puede no ser lo suficientemente potente en ciertos casos. Por ejemplo, imagine que un administrador quiere realizar una modificación sobre un Ítem porque detectó al menos una vez que en el metadato “dcterms.abstract” la palabra

“creación” estaba escrita “creasión”, para corregir este error el administrador puede ejecutar la siguiente consulta:

transformar:item(handle=HandleDeEjemplo - dcterms.abstract;creasión;creación)

Esta consulta reemplaza todas las ocurrencias de la palabra “creasión” por “creación” en el metadato dcterms.abstract y si bien esta consulta resolvería el problema planteado ¿qué ocurre si en el mismo metadato aparece la palabra “Creasión”? es decir se repite el mismo error pero la palabra en cuestión comienza con una mayúscula, en este caso la palabra no sería corregida, lo mismo ocurriría si la palabra estuviese escrita “cresion”, es decir sin tilde. Para abarcar todas las posibilidades el administrador debería escribir la siguiente consulta:

transformar:item(handle=HandleDeEJemplo - dcterms.abstract;creasión;creación , dcterms.abstract;Creasión;creación, dcterms.abstract;Creasion;creación, dcterms.abstract;creasion;creación)

Como se puede apreciar la escritura de la consulta se vuelve larga, repetitiva y tediosa. Es por este motivo que se decidió permitir a los usuarios que utilicen expresiones regulares para poder realizar selecciones todavía más específicas, si bien su uso requiere cierto, pero no demasiado, aprendizaje le brinda a la herramienta una gran potencia y una nueva gama de consultas posibles. Para resolver el mismo problema planteado pero utilizando expresiones regulares la consulta a escribir es:

transformar:item(handle=HanldeDeEjemplo - dcterms.abstact;[cC]reasi[oó]n;creación)

Como puede apreciarse el uso de las expresiones regulares no modifica la estructura de la consulta, simplemente se utiliza en el mismo lugar donde se indica la parte del metadato a modificar, esto permite que los administradores no deban aprender diferentes estructuras ni memorizar qué estructura va con qué tipo de consulta. En la solución planteada la expresión regular utilizada es: “*[cC]reasi[oó]n*”, en este caso dentro de los corchetes se encuentran las posibles variantes que puede presentar la palabra, es decir:

- Empezar con mayuscula o minuscula
- Tener tilde o no en la letra ‘o’

Al utilizar expresiones regulares la consulta se vuelve considerablemente más corta, se evitan las repeticiones y la complejidad que conlleva escribirla no aumenta a niveles que un usuario no informático no pueda entender. Por otro lado utilizar expresiones regulares permite realizar selecciones mucho más específicas y potentes lo cual puede prevenir la realización de modificaciones indeseadas, como por ejemplo podría ocurrir con una condición ambigua o no lo suficientemente restrictiva

Tener una vista anticipada

Cuando los administradores ejecuten consultas de transformación en la herramienta desarrollada en esta tesina les va a ser complicado saber anticipadamente cuál va a ser el resultado exacto de dicha ejecución. Esto ocurre porque incluso si la consulta solo afecta a una pequeña cantidad de elementos, la consulta puede tener un error de sintaxis que afecte

el resultado final por ejemplo si se desea corregir la palabra “ejempol” por “ejemplo” en un metadato, por ejemplo dc.title, de un Ítem la consulta a escribir sería la siguiente:

transformar:item(handle=HandleDeEjemplo - dc.title;ejempol;ejemplo)

Pero incluso en este sencillo ejemplo un administrador podría equivocarse al momento de escribir el handle, el metadato, el valor que quiere modificar o el nuevo valor que quiere poner. Como puede verse incluso en pequeñas consultas se pueden encontrar diversos lugares donde existe la posibilidad de una equivocación que cause un resultado final diferente al esperado. Si esto ocurre en una consulta como la del ejemplo, que solo modifica un Ítem, el posible problema no sea muy grave y la solución rápida y simple, pero siendo que las consultas que se pueden ejecutar con esta herramienta podrían afectar a todos los elementos del repositorio, en el caso de CIC-DIGITAL son más de 4000, resulta esencial que la herramienta presente un mecanismo que ayude a evitar este tipo de problemas.

Es por las razones expresadas en los párrafos precedentes que la herramienta presenta, ante cualquier consulta de transformación, un vista previa en la cual se le muestra al usuario el Handle del elemento a transformar, el Metadato que va a ser transformado, el valor antiguo del metadato (si corresponde) y el nuevo valor (si corresponde), con el objetivo de que el usuario pueda verificar si los cambios a realizar son los esperados, en cuyo caso deberá apretar un botón de confirmación o en caso contrario podrá modificar la consulta y tendrá nuevamente la posibilidad de revisar los resultados en la vista previa.

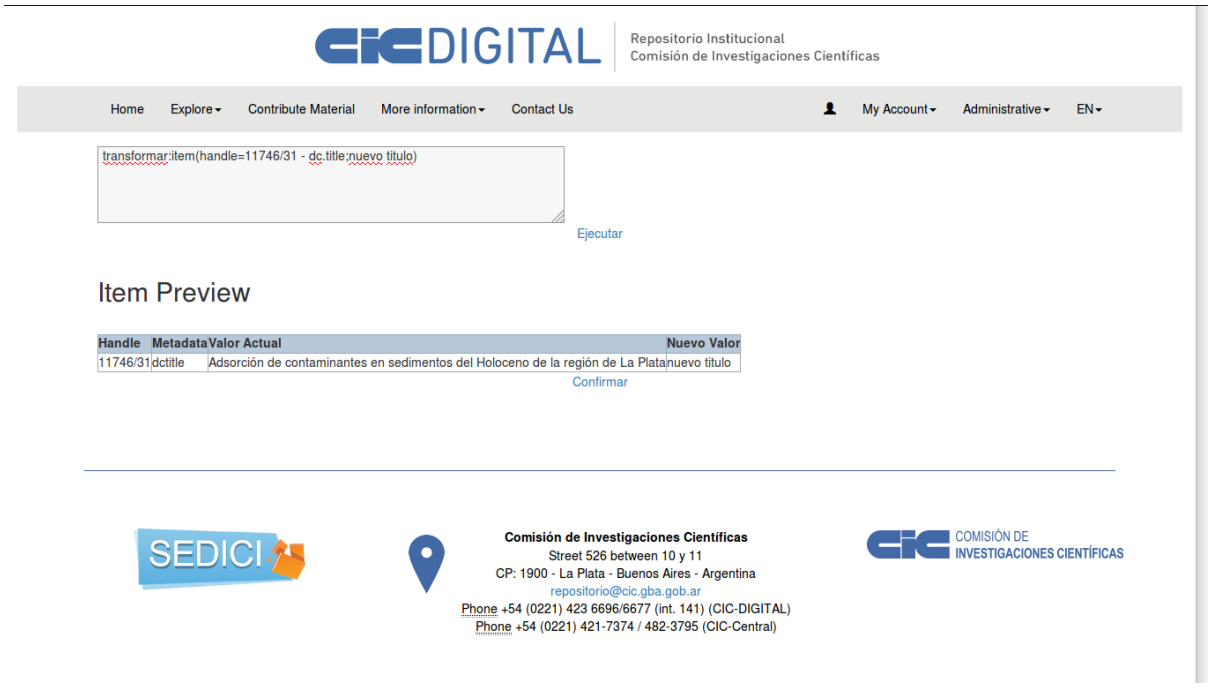


Figura 6.1: Se muestra la vista previa que genera la herramienta ante una consulta de transformación

En este capítulo se explicó el proceso de desarrollo del lenguaje de consulta que provee la herramienta desarrollada en esta tesina, comenzando por el uso que se le dió a la JSR-341, junto con el proceso de creación de la estructura de la sintaxis del lenguaje de consultas y a

su vez se explicó el uso de cada uno de los tipos de consultas que el usuario puede realizar, junto con la sintaxis elegida para cada uno de ellos, complementando con ejemplos concretos. En el siguiente capítulo se abordará la explicación de la implementación de la herramienta, para ello se explicará la lógica interna de la misma complementando con diagramas de modelado UML.

Capítulo 7 | Implementación de la herramienta

Introducción

A través de los distintos capítulos precedentes se explicó el estudio e investigación que se hizo sobre el estado del arte, luego se abordó el análisis sobre las características y requerimientos funcionales que debía cumplir la herramienta y en capítulo precedente se explicó el desarrollo e implementación de la sintaxis del lenguaje de consultas. En este capítulo se explicará el desarrollo de la herramienta haciendo foco en los puntos principales que componen su estructura interna, la explicación de cada una de las partes se centrará en su objetivo y funcionamiento complementado con un diagrama UML en el cual se mostrará las clases que la componen.

Módulo de Expresiones

Cuando un administrador ejecuta una consulta en la herramienta, lo primero que debe hacerse es crear una instancia de la clase “ELProcessor” y definir todas los nombres claves para cada uno de los métodos que el usuario puede ejecutar (la funcionalidad que provee la clase “ELProcessor” y la forma de asignar nombres claves a los métodos fue explicado en el capítulo 6 sección “Utilización de la JSR-341”). Sin embargo resulta poco eficiente pensar que estas acciones se realicen cada vez que un administrador quiera ejecutar una consulta, es por esta razón que se crearon las clases “ExpressionModule” y “ELInstancier”.

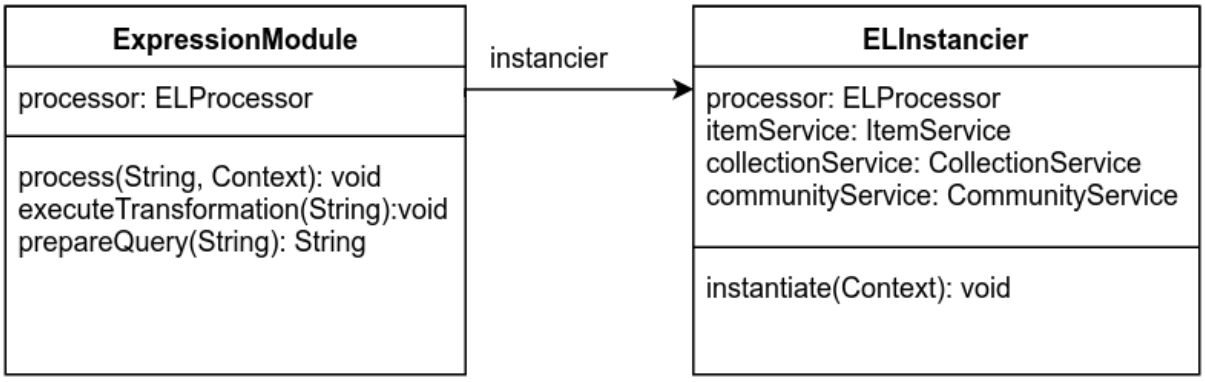


Figura 7.1: Se muestra un UML simplificado con las clases ExpressionModule y ELInstancier

La clase “ExpressionModule” es la encargada de recibir la petición del usuario, es decir que es la clase que inicia todo el proceso de interpretación de la consulta, y obtención y manipulación de los datos correspondientes. El primer paso en la interpretación de la

consulta es definir qué consulta es, es decir determinar cuál de las posibles consultas detalladas en el capítulo anterior fue escrita y sobre cuál DSO(Ítem, Colección o Comunidad) se quiere aplicar. Para realizar esta primera interpretación de la consulta se utiliza la JSR-341 junto con la sintaxis elegida, como fue explicado en el capítulo 6 sección “Creación y elección de la sintaxis”, la sintaxis elegida presenta la siguiente estructura:

accionARealizar:DSO(condiciones)

Como puede verse en el capítulo mencionado todas las posibles consultas que se pueden realizar con esta herramienta poseen la misma estructura, en base a este hecho se utilizó la funcionalidad de asignar nombres claves a métodos estáticos de una clase Java de la JSR-341 para que la combinación de “accionARealizar” y “DSO” de las consultas se correspondan con un método estático de una clase Java, la funcionalidad mencionada de la JSR-341 fue explicada con más detalle en el capítulo 6 sección “Utilización de la JSR-341”. Para clarificar lo expresado anteriormente se dará un pequeño ejemplo, la consulta presentada a continuación obtiene el Ítem con handle igual a 11746/5148

seleccionar:item(handle=11746/5148)

Siguiendo el hilo de pensamiento del párrafo anterior, el nombre clave de la consulta presentada es “seleccionar:item”, es decir que la JSR-341 se deberá configurar para que asigne a un método estático de una clase Java el nombre clave mencionado. La clase “ELInstancier” es la encargada de realizar dicha configuración, por lo tanto para realizar la primera interpretación de la consulta lo primero que debe hacer la clase “ExpressionModule” es indicarle a la clase “ELInstancier” que configure la JSR-341, para realizarla se utiliza una instancia de la clase “ELProcessor” que provee la JSR-341 (el funcionamiento de esta clase fue explicado con más detalle en el capítulo 6 sección “Utilización de la JSR-341”). Como se dijo anteriormente realizar esta configuración cada vez que un usuario quiera ejecutar una consulta resulta ineficiente, es por esta razón que la instancia de la clase “ELProcessor” es guardada en una variable estática, es decir que el tiempo de vida de la variable se extiende durante toda la ejecución del programa, de esta manera la instanciación y configuración de la clase “ELProcessor” solo se realiza una vez y luego simplemente se accede a ella. Una vez que la clase “ELInsancier” configura la JSR-341, la clase “ExpressionModule” debe decirle a través de la clase “ELProcessor” que evalúe la consulta que el usuario ingresó, esta evaluación (gracias a la configuración previamente realizada) desembocará en la ejecución de un método estático de una de dos clases Java, estas clases son llamadas “Actions”.

Actions

Las clases “Actions” son el punto de entrada al “core” o núcleo de la herramienta desarrollada en esta tesina, existen 2 clases “Actions”:

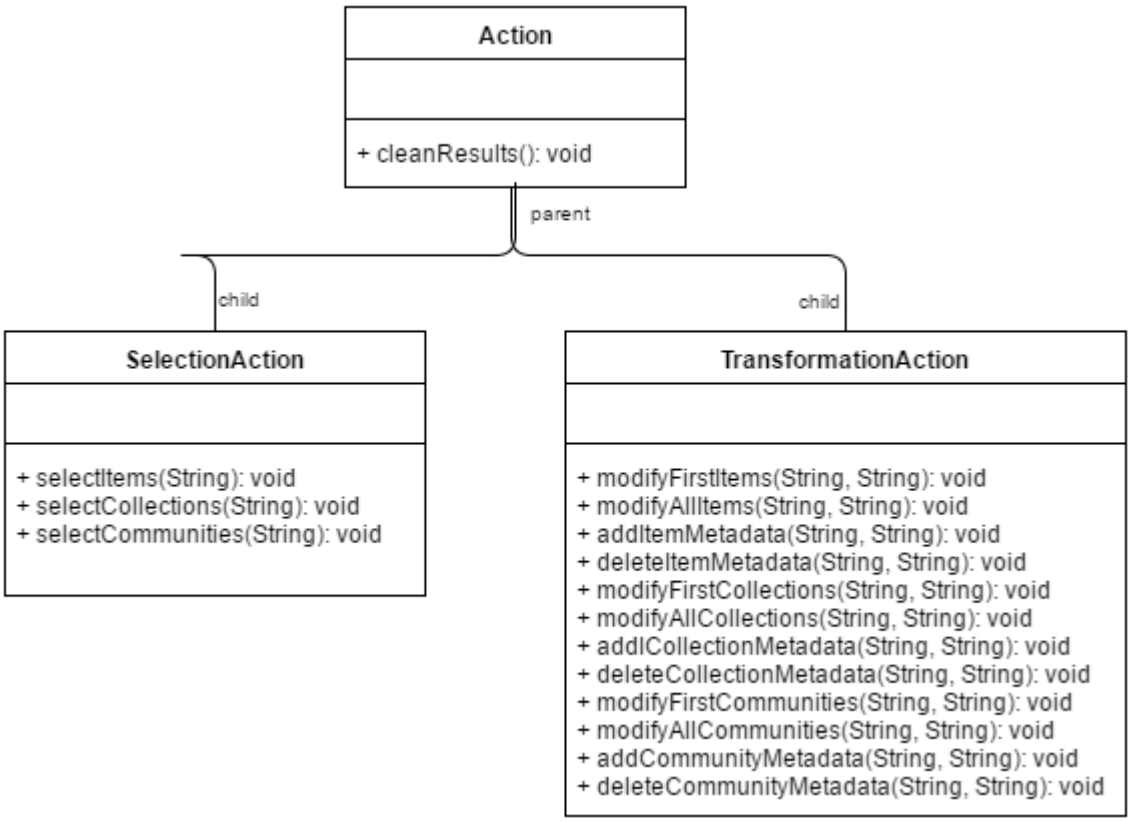


Figura 7.2: Se muestra un diagrama UML simplificado de las clases “Action” de la herramienta

Como puede interpretarse por los nombres de las clases, existe una clase “Action” para cada tipo de consulta que puede realizarse con la herramienta, la clase “SelectionAction” será ejecutada cuando el usuario realice una consulta de selección, mientras que la clase “TransformationAction” será ejecutada cuando se realice una consulta de transformación. Cabe aclarar que como las consultas de validación forman parte de las consultas de selección y transformación no fue implementada una clase “Action” para este tipo de consulta. El objetivo y la razón de creación de estas clases viene dado por una característica que exige la JSR-341, como se dijo previamente para que la JSR-341 pueda vincular un método Java con un nombre clave este método debe ser estático, y con el objetivo de utilizar clases y métodos de instancia en el “core” de la aplicación se decidió implementar estos dos puntos de entradas con métodos estáticos para que sean referenciados por la JSR-341. Asimismo estas clases tienen también como objetivo conectarse con el “core” de la herramienta, es decir conectarse con las clases que hacen el procesamiento fino sobre la consulta escrita por el administrador, estas clases son llamadas “Resolvers”.

Resolvers

Los “Resolvers” son los encargados de realizar el análisis e interpretación minuciosa de la consulta escrita por el usuario, entre otras cosas la interpretación conlleva determinar las condiciones que utiliza y si estan bien escritas, a su vez son los encargados de preparar la consulta que se ejecutará a nivel de base de datos para obtener o transformar (según corresponda) los elementos necesarios. Existen distintos “Resolvers” en la herramienta,

cada uno se encarga de una tarea en particular, esto permite tener el código agrupado y encapsulado lógicamente:

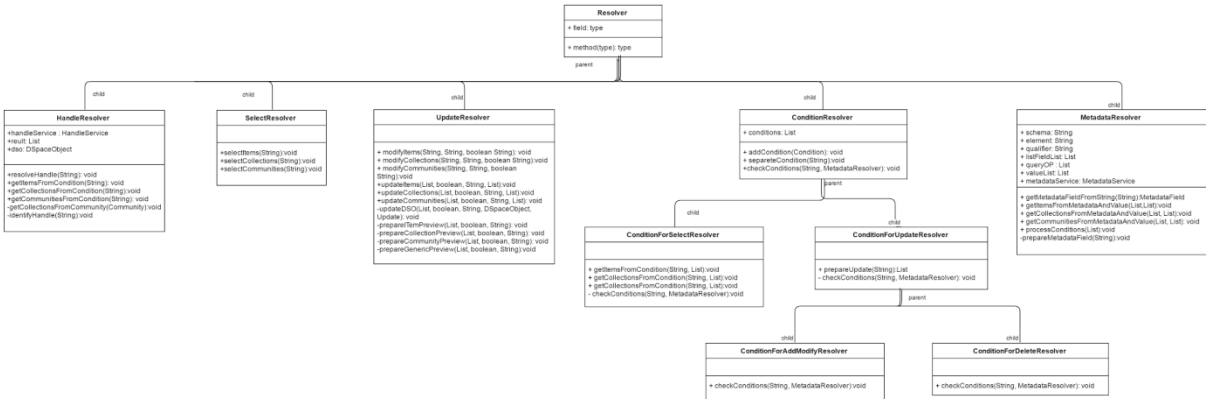


Figura 7.3: Se muestra un diagrama UML simplificado de las clases “Resovlers” que provee la herramienta

Como puede verse en la figura 7.3 existen diversos “Resolvers” dentro de la herramienta, en esta sección se dará un resumen del objetivo y funcionamiento de cada uno de ellos.

SelectResolver

Este resolver es el encargado de iniciar el proceso de interpretación de las condiciones de selección de la consulta, es decir que el “SelectResolver” solo se encarga de interpretar qué elementos desea recuperar el administrador. La razón principal por la cual se creò el “SelectResolver” es que permite reutilizar código, dado que si el usuario final escribe una consulta de selección este Resolver será el encargado de comenzar la interpretación de las condiciones que escribió el usuario para poder recuperar y luego mostrarle los elementos solicitados, a su vez si el usuario final escribe una consulta de Transformación dicho Resolver será utilizado para obtener los elementos que se desean transformar.

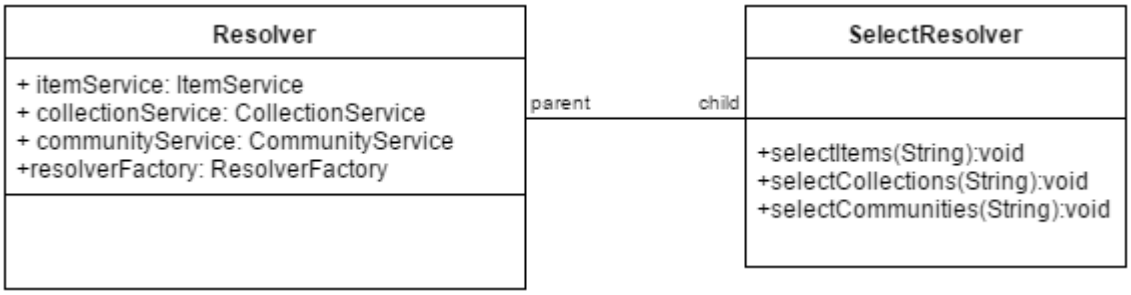


Figura 7.4: Se muestra un diagrama UML simplificado de la clase “SelectResolver” y su padre, la clase “Resolver”

HandleResolver

Como se dijo anteriormente el “SelectResolver” es el encargado de iniciar el proceso de interpretación de las condiciones de selección que el usuario final puso en la consulta, en otras palabras es el encargado de iniciar la obtención de los elementos que cumplan con las condiciones indicadas. Para esto se deben interpretar y verificar dichas condiciones, para realizar estas tareas de forma ordenada y lo más eficientemente posible se dividieron las condiciones de selección en dos categorías:

- Condición por handle
- Condición por validación

En esta sección se explicará la selección por handle, de la cual se encarga la clase “HandleResolver”:

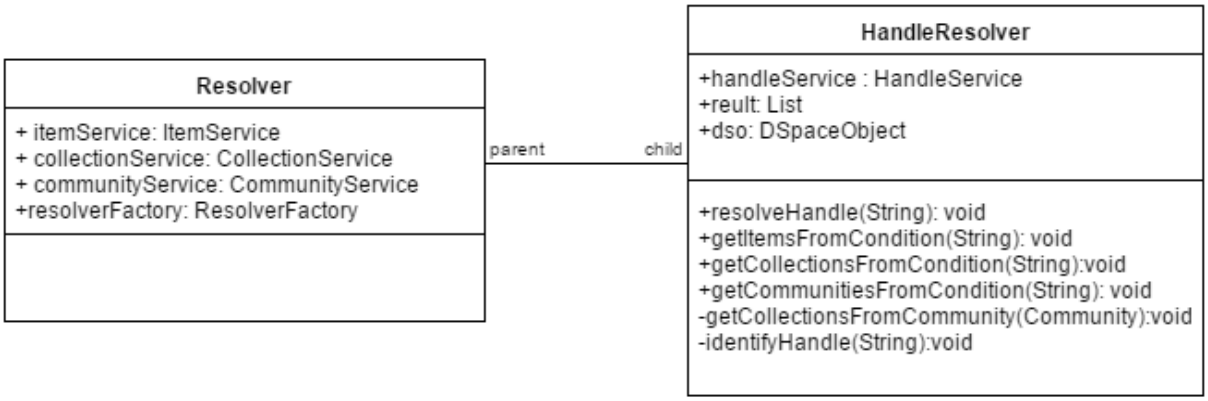


Figura 7.5: Se muestra un diagrama UML simplificado de la clase “HandleResolver” y su clase padre “Resolver”.

Como fue expresado anteriormente esta clase en la encargada de resolver las condiciones de selección por handle que se encuentren en la consulta, para ello la clase dispone de 3 métodos principales:

- getItemsFromCondition
- getCollectionsFromCondition
- getCommunitiesFromCondition

Estos métodos le permiten a esta clase identificar si debe recuperar un Ítem, una Colección o una Comunidad. En base a esa información y utilizando un método auxiliar llamado “resolveHandle”, el cual recupera un elemento en base a su Handle utilizando mecanismos provistos por DSpace, la clase sabe cómo debe actuar. Por ejemplo si se está ejecutando el método “getItemsFromCondition” y el método “resolveHandle” le indica que el handle es de un Ítem, la clase sabrá que el administrador quiere obtener ese Ítem, pero si el método “resolveHandle” le indica que el handle es de una Colección, la clase sabrá que el usuario final quiere recuperar los Ítems dentro de esa Colección. Es de esta manera que combinando uno de los 3 métodos principales más el método “resolveHandle” la herramienta identifica los elementos que el administrador desea recuperar.

ConditionForValidateResolver

Como se mencionó en el capítulo anterior sección “Consulta de Validación” un administrador tiene una amplia gama de posibles condiciones de validación que puede

agregar al momento de seleccionar elementos. La clase encargada de interpretar dichas condiciones de validación es “ConditionForValidateResolver”:

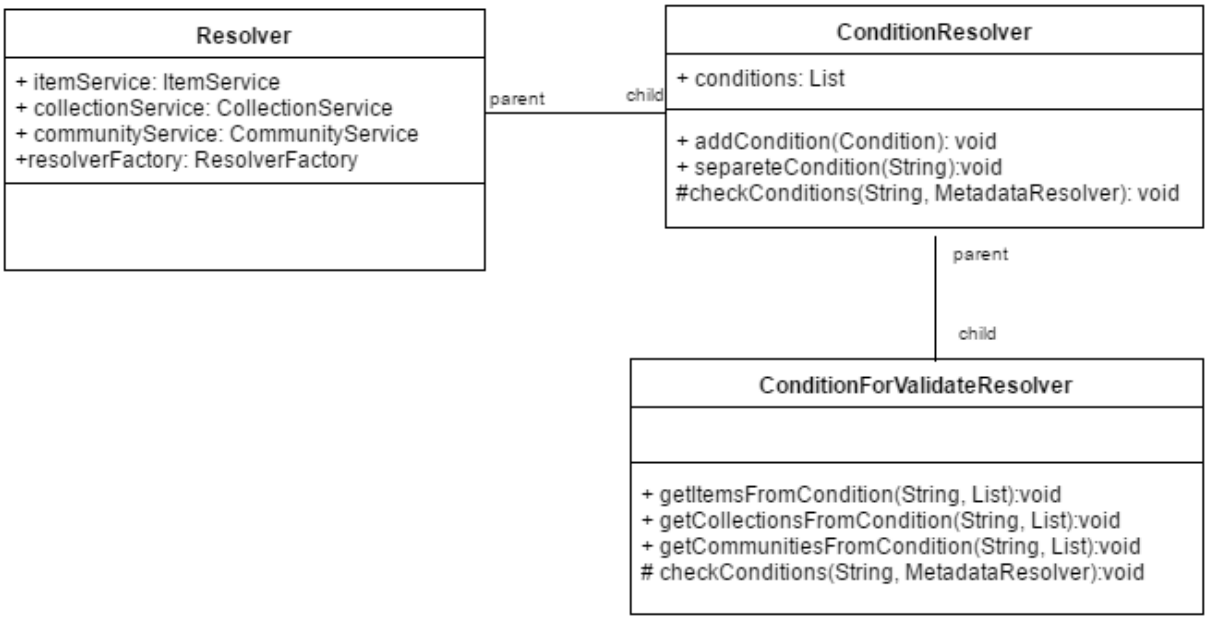


Figura 7.6: Se muestra un diagrama UML simplificado de la clase “ConditionForValidateResolver” junto con su clase padre “ConditionResolver” y su clase padre “Resolver”.

Este clase es la encargada de comenzar la interpretación de las condiciones de validación de la consulta y a su vez preparar la creación de la consulta que se hará a nivel de base de datos para obtener los elementos deseados. Para esto la clase cuenta con 3 métodos principales:

- getItemsFromCondition
- getCollectionsFromCondition
- getCommunitiesFromCondition

Los 3 métodos mencionados tienen un comportamiento similar, la primera acción que realizan es separar todas las condiciones de validación que existen en la consulta y para ello utilizan un método llamado “separateConditions”, el cual se encuentra definido en la clase padre “ConditionResolver” con el objetivo de rehusar código, una vez que las condiciones fueron separadas se utiliza el método “checkConditions” para identificar cada una de las condiciones de validación, por cada una de las condiciones identificadas se creará una instancia de la clase “Condition” la cual es una clase propia de la herramienta que funciona simplemente como almacenamiento de información. Una vez identificadas todas las condiciones de validación y creadas todas las instancias de la clase “Condition” correspondientes, se comienza a ejecutar la clase “MetadataResolver” la cual se encarga de la creación de la consulta de selección que se hará a nivel de base de datos.

MetadataResolver

Como se dijo anteriormente esta clase tiene como objetivo principal crear la consulta que se ejecutará a nivel de base de datos, esta consulta puede ser una consulta de selección. Además esta clase tiene como objetivo realizar los chequeos correspondientes a

los metadatos, es decir esta clase deberá corroborar que los metadatos ingresados por el administrador existan y tengan el formato correcto, dicho formato en el caso de CIC-DIGITAL es el formato DublinCore (Dublincore.org. 2017): *schema.element.qualifier* siendo "*qualifier"* opcional.

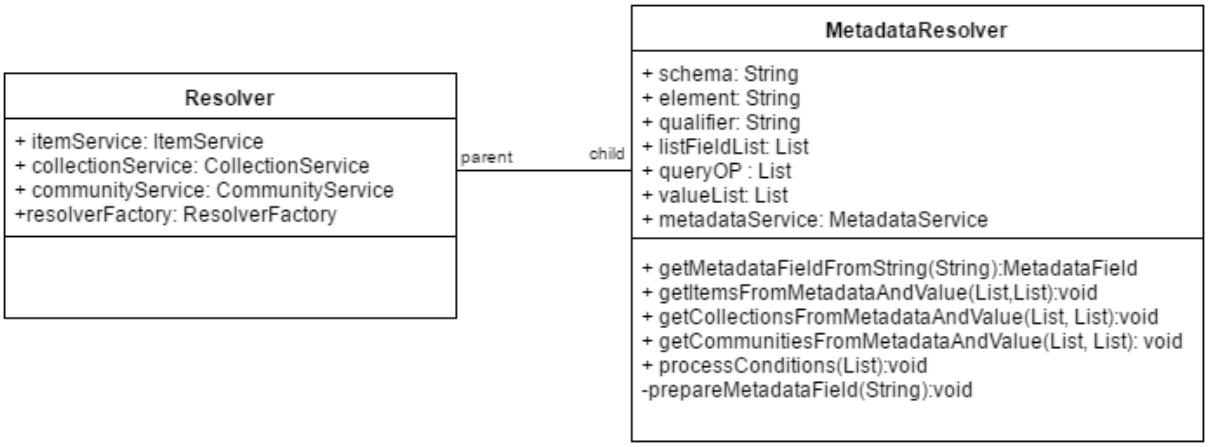


Figura 7.7: Se muestra un diagrama UML simplificado de la clase "MetadataResolver" y su clase padre "Resolver"

Como fue mencionado anteriormente uno de los objetivos de la clase "MetadataResolver" es verificar que los nombres de los metadatos ingresados por el usuario final sean correctos y además preparar la consulta de selección que se ejecutará a nivel de base de datos. Para realizar este primer objetivo la clase cuenta con un método llamado "getMetadataFieldFromString", este método utiliza una clase de DSpace llamada "MetadataField" la cual es una representación de un metadato dentro de DSpace, para verificar que los metadatos ingresados por el usuario existan en el dominio de CIC-DIGITAL se utiliza una funcionalidad de la clase "MetadataField" la cual permite verificar esta condición, en caso de que el metadato ingresado no exista dentro del domino de CIC-DIGITAL se le debe avisar al usuario de la situación. Por otro lado para poder cumplir el objetivo de preparar la consulta de selección que se ejecutará a nivel de base de datos, la herramienta cuenta con 3 métodos principales:

- getItemsFromMetadataAndValue
- getCollectionsFromMetadataAndValue
- getCommunitiesFromMetadataAndValue

Dependiendo del DSO que el administrador desea recuperar es el método que se ejecutará, cada uno de estos métodos utilizan un método auxiliar llamado "processConditions" el cual solo toma cada uno de los objetos "Condition" mencionados anteriormente y recupera la información que cada uno almacena, una vez hecho esto la clase "MetadataResolver" se comunica con un método propio de DSpace y le envía la información obtenida en un formato específico para que DSpace se encargue de recuperar los elementos que cumplan con todas las condiciones establecidas por el usuario.

De esta manera es como la herramienta realiza el proceso de interpretación de la consulta de selección y recupera los elementos correspondientes, a continuación se explicará como la herramienta realiza el proceso de transformación de los elementos.

UpdateResolver

Así como el “SelectResolver” es el encargado de iniciar el proceso de obtención de los objetos solicitados por el usuario, el “UpdateResolver” es el encargado de iniciar el proceso de transformación los objetos seleccionados y a su vez es el encargado de iniciar la preparación las vistas previas, las cuales tienen como objetivo evitar posibles errores involuntarios de los usuarios al momento de escribir una consulta proveyendo un listado de los elementos a modificar junto con las modificaciones que van a realizarse sobre cada uno de ellos.

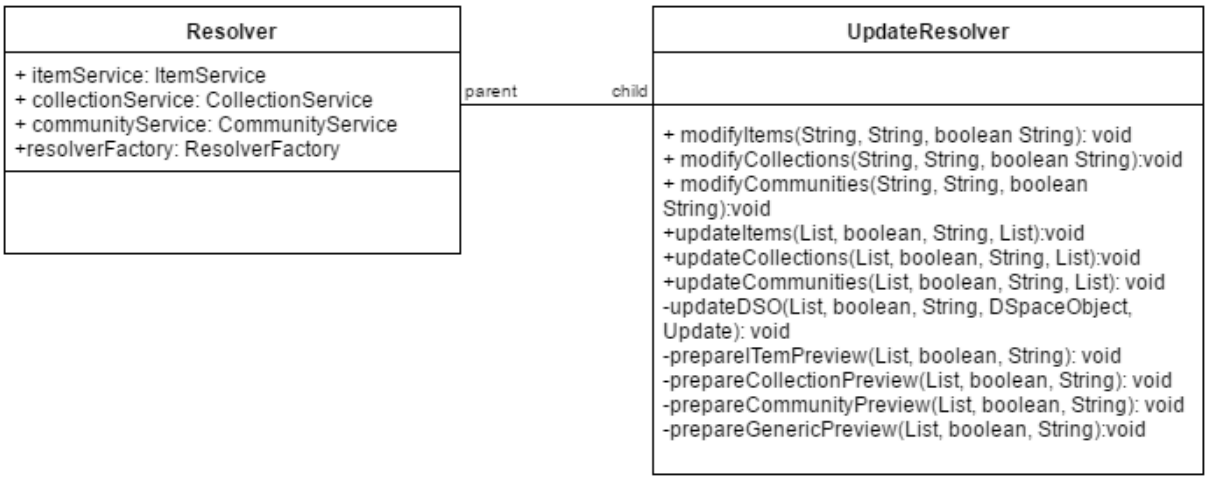


Figura 7.8: Se muestra una diagrama UML simplificado de la clase “UpdateResolver” y su padre, la clase “Resolver”

Como se dijo previamente el “UpdateResolver” tiene 2 objetivos, iniciar el proceso de preparado de las vistas previas e iniciar el proceso de transformación de los elementos seleccionados. Para realizar el segundo objetivo la clase posee 4 métodos:

- updateItem
- updateCollection
- updateCommunity
- updateDSO

Estos métodos poseen una lógica interna muy simple, dado que solo le indican a las clases “Update” (sobre las cuales se hablará más adelante en este capítulo) qué elementos deben modificar, qué tipo de modificación deben realizar sobre cada uno de ellos y también las modificaciones que desea realizar el usuario. Uno de los 4 método mencionados es el “updateDSO”, este método contiene código genérico que es reutilizado por los otros 3 métodos mencionados, lo cual hace al código más fácil de mantener y lo vuelve más claro y simple de entender. Para cumplir con el segundo objetivo, la clase “UpdateResolver” dispone de 7 métodos los cuales son:

- modifyItems
- modifyCollections
- modifyICommunities
- prepareItemPreview
- prepareCollectionPreview
- prepareCommunityPreview

- prepareGenericPreview

El objetivo de estos 7 métodos es el de iniciar el proceso de creación de las vistas anticipadas, para realizar esta acción hay varias sub tareas que se deben realizar y es por esta razón que estos 7 métodos se dividen en 2 subgrupos. El primero está compuesto por las clases:

- modifiyItems
- modifyCollections
- modifyCommunities

El objetivo de estas clases es, en primer lugar, comunicarse con la clase “SelectResolver” para que ésta inicie el proceso de obtención de los elementos a transformar, luego deben iniciar el proceso de interpretación de las condiciones de modificación que el administrador puso en la consulta, para esto se utiliza un método auxiliar llamado “prepareConditions” el cual se comunicaŕa con las clases “ConditionForAddModifyResolver” y “ConditionForDeleteResolver” que son las encargadas de identificar las condiciones de transformación y por cada una de ellas crearán un objeto “Condition” el cual, como se dijo anteriormente, funciona como un simple contenedor de información. En este punto ya se tiene como información que elementos se quieren transformar y qué transformaciones se quiere realizar sobre los mismos, por lo tanto entran en juego los 4 métodos restantes. Los ya mencionados métodos tienen como objetivo tomar la información obtenida y comenzar el proceso de identificación de las transformaciones, es decir que se debe comenzar a identificar qué transformaciones se harán efectivamente sobre cada elemento seleccionado, para así poder mostrarle al usuario el elemento a transformar, el metadato que se quiere transformar, el valor actual del metadato y el valor final del mismo (luego de aplicada la transformación). Para realizar este proceso los 4 métodos mencionados utilizan dos módulos de la herramienta desarrollada en esta tesina, los cuales son:

- Updates
- Managers

ConditionForUpdateResolver

Antes de continuar con el siguiente módulo de la herramienta, es necesario explicar el funcionamiento de 2 “Resolvers” más, cuyo objetivo es el de identificar las condiciones de transformación que existan en la consulta y crear un objeto “Condition” por cada una de ellas.

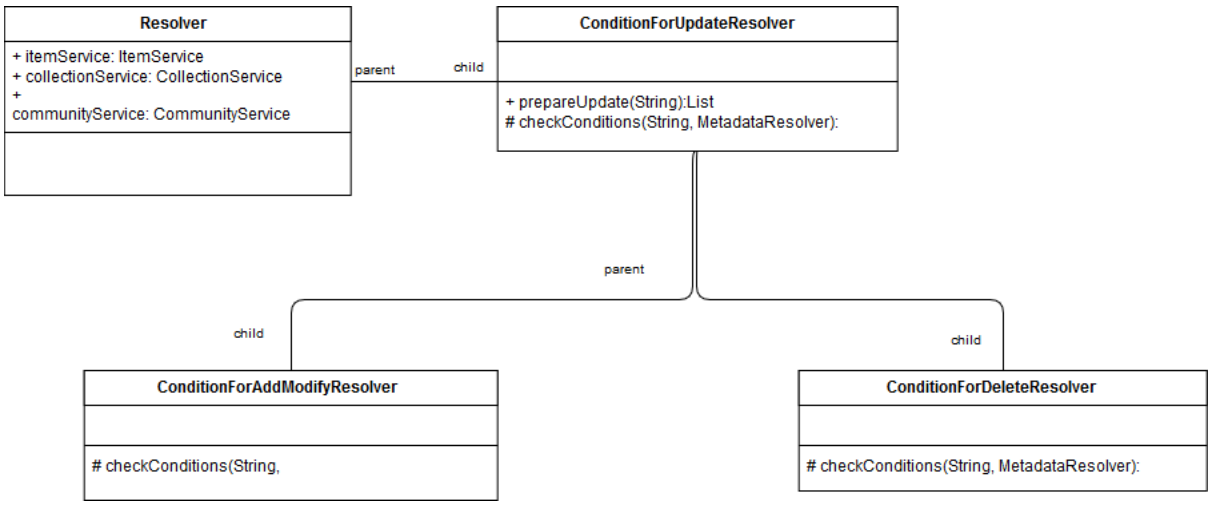


Figura 7.6: Se muestra un diagrama UML simplificado de la clase “ConditionForUpdateResolver” junto con su clase padre “Revsolver” y sus dos clases hijas “ConditionForAddModifyResolver” y “ConditionForDeleteResolver”

Como puede verse en la figura 7.6 existe una clase llamada “ConditionForUpdateResolver” la cual tiene dos clases hijas:

- ConditionForAddModifyResolver
- ConditionForDeleteResolver

Cada una de estas subclases posee un solo método el cual se encarga de identificar las condiciones de transformación que existan en la consulta. A su vez la clase “ConditionForUpdateResolver” posee código genérico que es utilizado por ambas clases hijas, lo que permite rehusarlo y no repetirlo en cada una de ellas.

Updates

Las clases “Updates” son las encargadas de cumplir 2 objetivos, identificar efectivamente (en base a la información previamente obtenida) que va a ser transformado en cada elemento seleccionado y realizar dicha transformación.

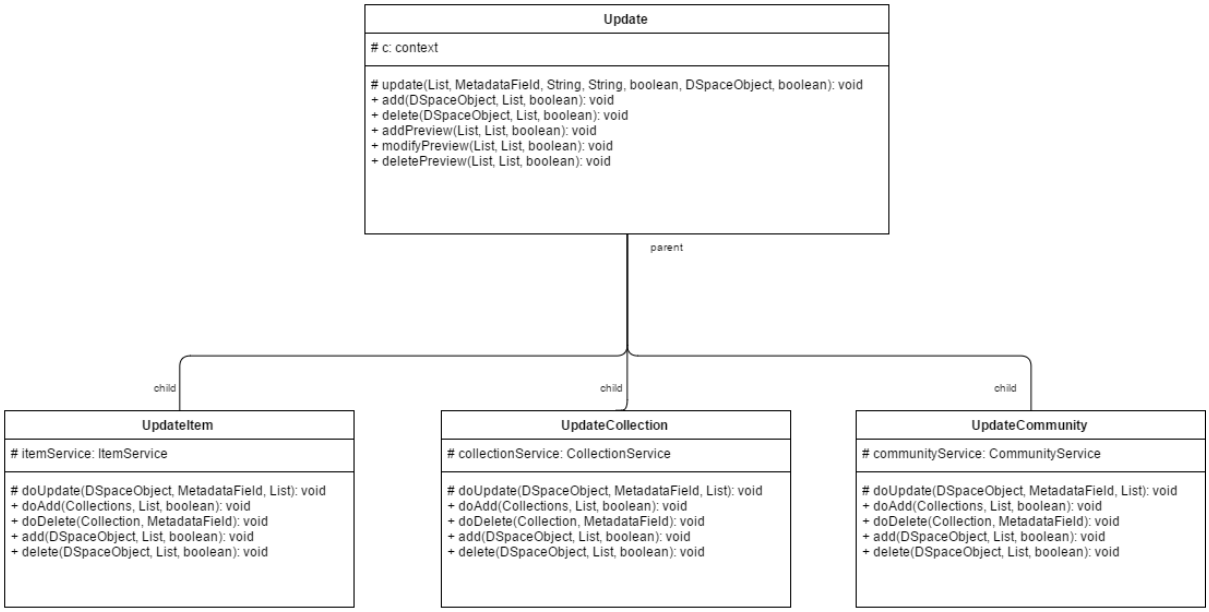


Figura 7.9: Se muestra un diagrama UML simplificado de la clases “UpdateItem”, “UpdateCollection” y “UpdateCommunity” junto con su clase padre “Update”

Como puede verse en la figura 7.9, existen 4 clases “Updates” en la herramienta desarrollada en esta tesina: La clase “Update” la cual es la clase padre de las restantes, esta clase contiene todo el código genérico es decir que contiene todo el código que puede ser utilizado por sus 3 clases hijas, de esta forma se reutiliza código lo cual hace que el mismo sea más limpio y fácil de mantener. Luego existen 3 clases hijas de la clase “Update”: “UpdateItem”, “UpdateCollection” y “UpdateCommunity”. Cada una de estas clases cumple un rol similar, con la diferencia de que cada una lo cumple para un DSO en particular, como puede verse en el diagrama las clases cuentan con 6 métodos principales:

- doUpdate
- doAdd
- doDelete
- modifyPreview
- addPrivew
- deletePreview

Los últimos 3 métodos mencionados son los encargados de obtener la información que se mostrará en los “preview” o vista anticipada de los cambios, es decir que estos métodos se ejecutarán cuando el administrador escriba una consulta en la herramienta y la ejecute. El objetivo de estos métodos es el de utilizar la información que le brinda el “UpdateResolver” e identificar qué cambios se realizarán sobre cada elementos, para con esa información mostrarle al usuario la vista anticipada de los cambios, para que él pueda decidir si desea o no ejecutar la consulta. Como puede verse estos 3 métodos no se encuentran definidos en ninguna de las clases hijas, sino que están definidos e implementados solo en la clase padre, esto implica que el código que se ejecuta para todas las vistan anticipadas (sin importar si los elementos a transformar son Ítems, Colecciones o Comunidades) es genérico por lo que facilita su mantenimiento. Por otro lado, los 3 primeros métodos mencionados son los encargados de realizar las transformaciones, es decir que estos métodos se ejecutarán una vez que el administrador haya escrito la consulta, visto la vista anticipada de los cambios y haya decidido realizar efectivas las

transformaciones, dependendiendo de los elementos que se vayan a modificar (Ítems, Colecciones, Comunidades) es la clases que se ejecutará y dependiendo de la transformación que se tenga que efectuar (agregar, eliminar o modificar un metadato) es el método que se ejecutará.

Managers

Existen 2 clases “Managers” en la herramienta desarrollada en esta tesina, ambas clases son *estáticas*, lo que significa que no se deben instanciar para poder ser utilizadas, lo cual permite un fácil acceso a ellas desde cualquier punto de la herramienta.

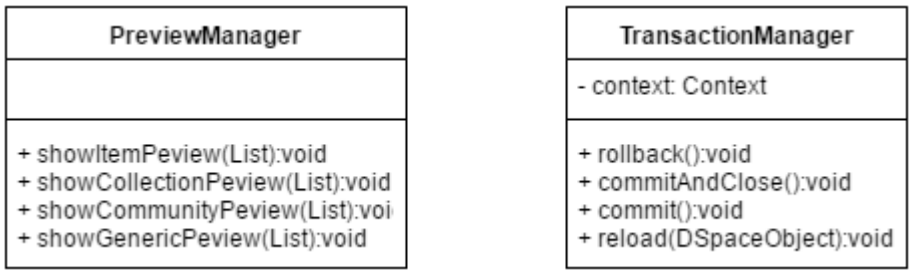


Figura 7.10: Se muestra un diagrama UML simplificado de las clases “TransactionManager” y “PreviewManager”

La clase “PreviewManager” es la encargada de brindar toda la información correspondiente a la vista anticipada, es decir que da acceso a toda la información que se obtuvo de la ejecución de los “Resolver” y de los “Updates”. Como puede verse en la figura 7.10 la clase consta de 4 métodos, uno por cada elemento que se puede transformar y un método genérico el cual contiene código reusable por los 3 métodos ya mencionados.

Como se dijo en el capítulo 5 sección “Requerimiento Funcionales” subsección “Retroceso automático”, la herramienta debe proveer un mecanismo de retroceso automático en caso de un fallo en medio de la ejecución de una consulta lo que evitaría, entre otras cosas, inconsistencias en el sistema. Para realizar esta acción se creó la clase “TransactionManager” la cual es la encargada de manejar la transacción de la consulta, es decir que esta clase es la encargada de retroceder todos los cambios en caso de que falle la ejecución de la consulta, como así también es la responsable de persistir todos los cambios en caso de que la ejecución de la consulta no falle.

Como se explico a lo largo de este capítulo existen diversos módulos principales que componen la herramienta desarrollada en esta tesina, al momento de realizar la implementación de cada uno de ellos se tuvo en cuenta distintas buenas prácticas de la programación, entre las cuales podemos encontrar:

- Comenzar los nombres de las clases con mayúscula
- Comenzar los nombres de los métodos con minúscula
- Escribir los nombres de las clases, métodos y variables en ingles
- Reutilizar código

Una vez que el desarrollo de la herramienta estuvo terminado se probó el resultado final obtenido en el repositorio CIC-DIGITAL, lo cual era uno de los objetivos planteados en esta tesina, las pruebas y resultados obtenidos, entre otras cosas, serás expuestas y explicadas en el próximo capítulo.

Capítulo 8 | Ejemplos en producción, conclusión y trabajo futuros

En el presente capítulo se explicarán las distintas pruebas realizadas en el repositorio CIC-DIGITAL complementando la explicación con las capturas de pantallas correspondientes. De esta manera se intentarán abarcar todas las funcionalidades que provee la herramienta tal y como fueron explicadas en el capítulo 6 de esta tesina, luego se detallarán las distintas conclusiones obtenidas para finalizar con los trabajos futuros propuestos.

Ejemplos en producción

En esta sección del capítulo 8 se mostrarán consultas reales que fueron ejecutadas sobre el repositorio CIC-DIGITAL, para ello se da una explicación de la consulta a realizar, el resultado esperado de la misma y se complementa con distintas capturas de pantalla donde puede verse el resultado real de la ejecución de las consultas. Cabe aclarar que las consultas utilizadas en los posteriores ejemplos no resuelven problemas reales, sino que fueron utilizadas porque permiten demostrar las distintas funcionalidades que provee la herramienta desarrollada, sin embargo dichas consultas si fueron ejecutadas sobre elementos que se encuentran en el repositorio CIC-DIGITAL.

Consultas de selección

Seleccionar todos los elementos del repositorio

Como fue explicado anteriormente la herramienta permite realizar consultas de selección sobre los distintos elementos del sistema, para ello se ejecuta la siguiente consulta en la herramienta *seleccionar:item()*

Figura 8.1: Se muestra una consulta que devuelve todos los elementos del repositorio

Como puede verse en la figura 8.1 la herramienta presenta un elemento “HTML” denominado “textarea” el cual es un campo de tamaño ajustable en el cual el administrador puede escribir la consulta que desea ejecutar, también puede apreciarse el botón

denominado “Ejecutar” el cual ejecuta la consulta ingresada. A continuación se muestra una imagen con el resultado de la ejecución de la consulta.

Handle	Metadata	Valor Actual	Nuevo Valor
11746/31	dc.title	Adsorción de contaminantes en sedimentos del Holoceno de la región de La Plata	-
11746/78	dc.title	Caracterización de la durabilidad de hormigones con arenas de trituración	-
11746/12	dc.title	Resistencia a la corrosión por picado del acero inoxidable AISI 316L nitrurado iónicamente	-
11746/131	dc.title	Patrimonio religioso en colonias, pueblos y localidades de la provincia de Buenos Aires: Anales LEMIT. Serie III, año 1, nº 1	-
11746/128	dc.title	70º Aniversario del LEMIT 1942 - 2012	-
11746/45	dc.title	Encuesta de alimentación y actividad física: síntesis de resultados iniciales	-
11746/32	dc.title	Propiedades de agregados obtenidos por combinación de arena de río con agregados finos reciclados	-
11746/130	dc.title	Aplicación del Método Acelerado CAMBT para evaluar la reactividad alcalina de agregados	-
11746/23	dc.title	Informe científico de investigador: Ortale, María Susana (2010-2011)	-
11746/98	dc.title	Factores que afectan a la estructura de los recubrimientos de cinc obtenidos por inmersión	-
11746/30	dc.title	Determinación de la composición de morteros de la envolvente del Museo de Ciencias Naturales de la ciudad de La Plata	-
11746/8	dc.title	Estudios ópticos y fotoacústicos en materiales con transiciones orden - desorden	-
11746/132	dc.title	Nuevas tecnologías sustentables aplicadas a la pavimentación asfáltica: Anales LEMIT. Serie III, año 1, nº 3	-
11746/77	dc.title	Estudio de la reactividad alcalina potencial de algunas rocas dolomíticas de la Argentina frente a la reacción álcali-carbonato	-
11746/144	dc.title	Informe científico de investigador: D' Angelo, Cristian Adrián (2013)	-
11746/129	dc.title	Reacción Álcali – Sílice en hormigones con ceniza de cáscara de arroz residual	-
11746/68	dc.title	30 años (1977-2007): Centro de Investigaciones Ópticas (CIOp)	-
11746/97	dc.title	Efecto de la deformación por tracción sobre la orientación cristalográfica del recubrimiento y la resistencia a la corrosión en CNS de chapas de	-

Figura 8.2: Se muestra los resultados de ejecutar la consulta que selecciona todos los Ítems del repositorio

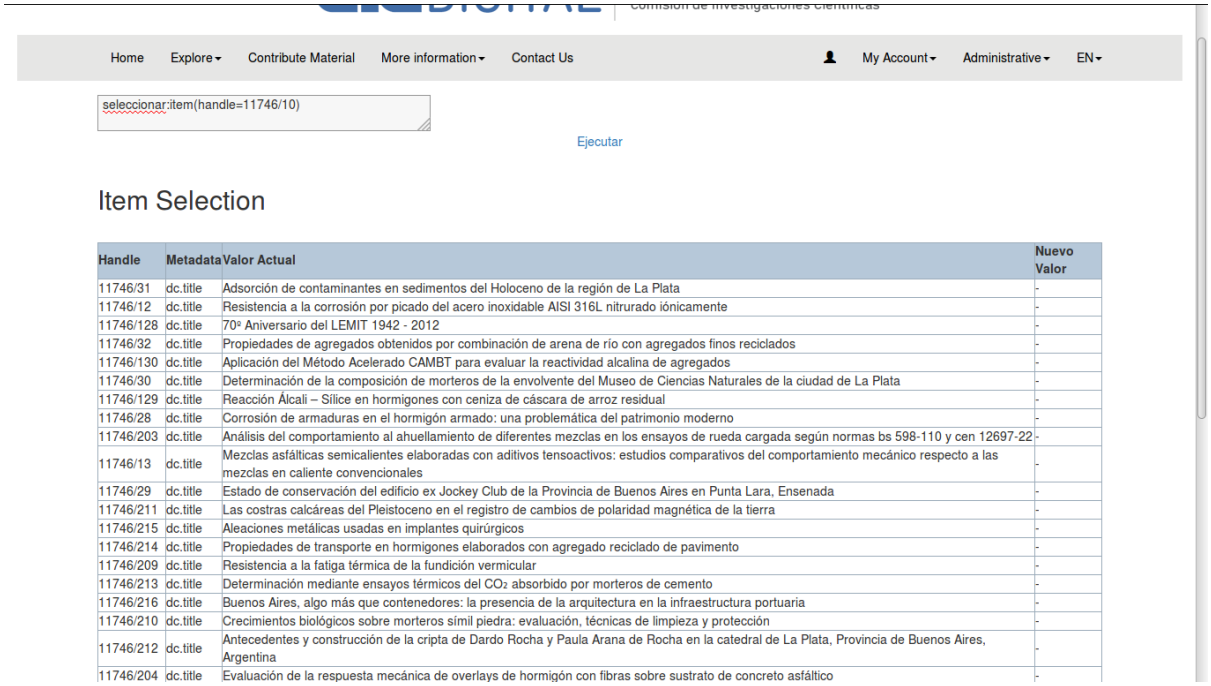
Como puede verse en la figura 8.2 al ejecutar una consulta de selección se despliega una tabla la cual posee 4 columnas:

- Handle del elemento
- Metadato
- Valor Actual
- Nuevo Valor

La primera columna contiene el valor del Handle del elemento, la segunda columna contiene un metadato del elemento, en las consultas de selección ese metadato es el título del elemento (el cual puede servir para identificarlo), mientras que en las consultas de transformación esa columna contiene el metadato a modificar. La tercera columna contiene el valor actual del metadato, este valor puede ser representado por un guión medio ‘-’ cuando no corresponda que se muestre ningún valor, por ejemplo en las consultas de agregación de metadatos. La cuarta columna contiene el nuevo valor del metadato, al igual que la columna anterior este valor puede ser representado con un guión medio ‘-’, por ejemplo en las consultas de selección o eliminación.

Selección por handle

A continuación se muestra la ejecución de una consulta de selección de Ítems que se encuentren dentro de una comunidad: *seleccionar:item(handle=11746/10)*



Handle	Metadata	Valor Actual	Nuevo Valor
11746/31	dc.title	Adsorción de contaminantes en sedimentos del Holoceno de la región de La Plata	-
11746/12	dc.title	Resistencia a la corrosión por picado del acero inoxidable AISI 316L nitrurado iónicamente	-
11746/128	dc.title	70º Aniversario del LEMIT 1942 - 2012	-
11746/32	dc.title	Propiedades de agregados obtenidos por combinación de arena de río con agregados finos reciclados	-
11746/130	dc.title	Aplicación del Método Acelerado CAMBT para evaluar la reactividad alcalina de agregados	-
11746/30	dc.title	Determinación de la composición de morteros de la envolvente del Museo de Ciencias Naturales de la ciudad de La Plata	-
11746/129	dc.title	Reacción Álcali – Sílice en hormigones con ceniza de cáscara de arroz residual	-
11746/28	dc.title	Corrosión de armaduras en el hormigón armado: una problemática del patrimonio moderno	-
11746/203	dc.title	Análisis del comportamiento al ahuellamiento de diferentes mezclas en los ensayos de rueda cargada según normas bs 598-110 y cen 12697-22	-
11746/13	dc.title	Mezclas asfálticas semicalientes elaboradas con aditivos tensoactivos: estudios comparativos del comportamiento mecánico respecto a las mezclas en caliente convencionales	-
11746/29	dc.title	Estado de conservación del edificio ex Jockey Club de la Provincia de Buenos Aires en Punta Lara, Ensenada	-
11746/211	dc.title	Las costras calcáreas del Pleistoceno en el registro de cambios de polaridad magnética de la tierra	-
11746/215	dc.title	Aleaciones metálicas usadas en implantes quirúrgicos	-
11746/214	dc.title	Propiedades de transporte en hormigones elaborados con agregado reciclado de pavimento	-
11746/209	dc.title	Resistencia a la fatiga térmica de la fundición vermicular	-
11746/213	dc.title	Determinación mediante ensayos térmicos del CO_2 absorbido por morteros de cemento	-
11746/216	dc.title	Buenos Aires, algo más que contenedores: la presencia de la arquitectura en la infraestructura portuaria	-
11746/210	dc.title	Crecimientos biológicos sobre morteros símil piedra: evaluación, técnicas de limpieza y protección	-
11746/212	dc.title	Antecedentes y construcción de la cripta de Dardo Rocha y Paula Arana de Rocha en la catedral de La Plata, Provincia de Buenos Aires, Argentina	-
11746/204	dc.title	Evaluación de la respuesta mecánica de overlays de hormigón con fibras sobre sustrato de concreto asfáltico	-

Figura 8.3: Se muestra la ejecución de una consulta de selección por el handle de la Comunidad padre

Como puede verse la consulta a ejecutar es similar a la de selección de todos los Ítems del repositorio, salvo por la diferencia de que la consulta ejecutada en la figura 8.3 se especificó el handle de una Comunidad para que la herramienta recupere todos los Ítems que pertenecieran a ella.

Selección por validación

A continuación se muestra la ejecución de una consulta de validación, la misma recupera todos los Ítems del sistema que cumplan con la condición de que su metadato “dcterms.creator.author” contenga (Like) a “Schinca”. Como fue expresado en el capítulo 6 en la sección “Consulta de validación” el carácter elegido para realizar la consulta de validación like es: ~ por lo tanto la consulta a ejecutar es la siguiente:

seleccionar:item(dcterms.creator.author ~ Schinca)

Handle	Metadata	Valor Actual	Nuevo Valor
11746/1229	dc.title	Alternative method for concentration retrieval in differential optical absorption spectroscopy atmospheric gas pollutant measurements	-
11746/3352	dc.title	Láser de monóxido de carbono	-
11746/178	dc.title	Informe científico de investigador: Schinca, Daniel Carlos (2011-2012)	-
11746/1231	dc.title	Excitation Mechanisms and Characterization of a Multi-Ionic Xenon Laser	-
11746/503	dc.title	Excited state absorption in KCl:EU^{2+}	-
11746/1904	dc.title	Informe científico de investigador: Schinca, Daniel Carlos (2013-2014)	-
11746/224	dc.title	Láser de monóxido de carbono. Estudio espectroscópico del sistema Angstrom	-
11746/479	dc.title	Alternative method for concentration retrieval in differential optical absorption spectroscopy atmospheric gas pollutant measurements	-
	dc.title	Excitation Mechanisms and Characterization of a Multi-Ionic Xenon Laser	-
11746/496	dc.title	Cleaning and characterization of objects of cultural value by laser ablation	-
	dc.title	Alternative method for concentration retrieval in differential optical absorption spectroscopy atmospheric gas pollutant measurements	-
11746/566	dc.title	Visible and near-infrared backscattering spectroscopy for sizing spherical microparticles	-
11746/557	dc.title	Spectroscopic study of population inversion mechanisms in diatomic-molecule lasers	-
11746/1992	dc.title	Population mechanisms in visible carbon monoxide pulsed lasers	-
11746/501	dc.title	Back isomerization from the excited state photoisomer of the laser dye 3,3' -diethyloxadicarbocyanine IODIDE (DODCI)	-
11746/1091	dc.title	Desarrollo de equipos ópticos para medir SO2 en chimeneas y aire ambiente	-
11746/558	dc.title	Excitation mechanisms and characterization of a multi-ionic xenon laser	-
11746/1902	dc.title	Laser spectroscopic analysis of N_2 pulsed discharges at low temperatures	-

Figura 8.4: Se muestran los items cuyo metadato dcterms.creator.author contiene la palabra "*Schinca*"

Consultas de Transformación

A continuación se mostrarán distintos ejemplos de consultas de transformación, para ello se explicará la transformación que se va a realizar, se mostrará el Ítem previo a dicha transformación, luego se mostrará la ejecución de la consulta para finalmente mostrar el estado final del Ítem.

Modificación de metadatos

En esta subsección se muestra la ejecución de una consulta de modificación, para ser más específico se muestra la ejecución de una consulta que seleccione todos los Ítems que contenga la Comunidad con handle "*11746/10*" y modificará aquellos cuyo metadato "dc.title" contenga el valor "*Adsorción*" reemplazando dicho valor por la palabra "ejemplo". En síntesis, la consulta a ejecutar es la siguiente:

transformar:item(handle=11746/10 - dc.title;Adsorción;ejemplo)

La consulta mencionada devolverá un solo resultado, es decir que dentro de los Ítems que se encuentran en la Comunidad con handle *11746/10*, llamada "Revista Ciencia y Tecnología de los Materiales", existe solo uno que posee en su título la palabra "Adsorción". En la figura 8.5 se ve ejecutada la consulta y se muestra el "preview" o vista anticipada que genera la herramienta, tal y como fue explicada en el capítulo 6 sección "Tener una vista anticipada". Luego se mostrará el estado del Ítem previo a la confirmación de la ejecución de la consulta, para finalmente mostrar el estado del Ítem posterior a la efectiva realización de la modificación.

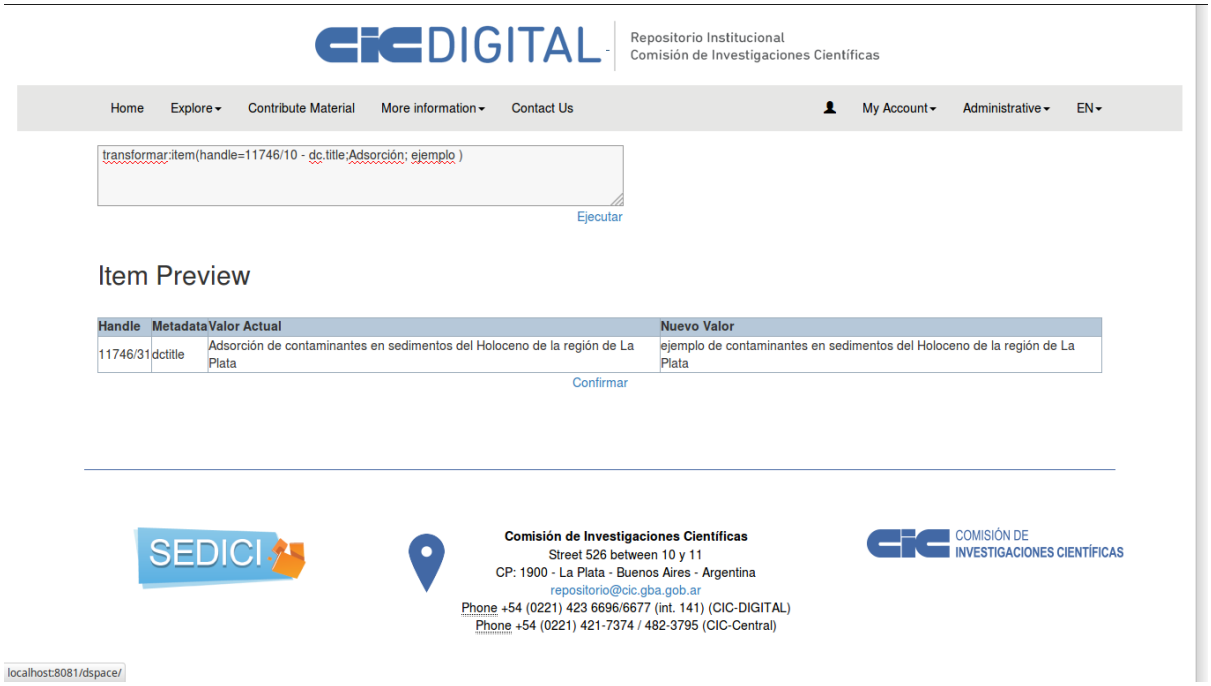


Figura 8.5: Se muestra la vista anticipada de la ejecución de la consulta mencionada anteriormente

Una vez que la consulta fue escrita por el administrador la herramienta le mostrará la vista anticipada o “preview”. Como puede verse en la figura 8.5 la vista anticipada es la misma tabla que se utiliza para las consultas de selección, esto permite mantener una misma base en las distintas vistas que se le pueden presentar al usuario, también puede verse que en la tabla en cuestión se encuentran todos los Ítems que se verán afectados por la consulta (en el caso del ejemplo es solo uno), para cada uno de los elementos se especifica su handle (en el caso del ejemplo es 11746/31), el metadato que va a ser modificado (en este caso dc.title), el valor actual del metadato (en el ejemplo dado es “ Adsorción de contaminantes en sedimentos del Holoceno de la región de La Plata”) y finalmente el nuevo valor (en este caso corresponde a “ejemplo de contaminantes en sedimentos del Holoceno de la región de La Plata”). También puede verse en la figura 8.5 que cuando la herramienta presenta la vista anticipada también agrega un botón de confirmación, el cual le permite al administrador hacer efectivos los cambios correspondientes. Cabe recordar que, como fue expresado anteriormente, esta funcionalidad le permite a los administradores corroborar que los cambios que van a ser realizados sean, efectivamente, los cambios deseados y de esta manera se intenta reducir la posibilidad de que ocurran errores involuntarios por parte de los administradores..

A continuación se presenta una imagen del estado del Ítem previo a la confirmación de la consulta en la cual se puede apreciar el título de Ítem:

Figura 8.6: Se muestra el Ítem con handle 11746/31 previo a la ejecución de la consulta

Como puede verse en la figura 8.6 el Ítem que se verá afectado por la consulta previamente mencionada tiene como título “Adsorción de contaminantes en sedimentos del Holoceno de la región de La Plata”. En la figura 8.7 puede verse el estado del mismo Ítem pero luego de la confirmación de la ejecución de la consulta:

Figura 8.7: Se muestra el estado del Ítem luego de la confirmación de la ejecución de la consulta

Agregación de metadatos

Para el primer ejemplo de esta funcionalidad se mostrará la ejecución de una consulta de agregación de metadato en la cual se seleccionará un Ítem por su handle y se le agregará el metadato “dcterms.subject” con el valor “Computing”. El Ítem tomado como ejemplo posee el handle 11746/3381 y título “ Integrando UML y DSL en el enfoque MDA ”, en la siguiente imagen se muestra dicho Ítem resaltando los valores que posee en el metadato “dcterms.subject” para que luego de efectuada la consulta sirva de punto de comparación.

Figura 8.8: Se muestra el estado del Ítem previo a la ejecución de la consulta de transformación

A continuación se presenta la consulta correspondiente para realizar la transformación mencionada anteriormente:

agregar:item(handle=11746/3381 - dcterms.subject;Computing)

Con esta consulta la herramienta recuperará el Ítem cuyo handle es 11746/3381 y le agregará el metadato “dcterms.subject” con valor “Computing”, en la siguiente figura se muestra la ejecución de la consulta junto con la vista previa que genera la herramienta para intentar evitar posibles errores involuntarios.

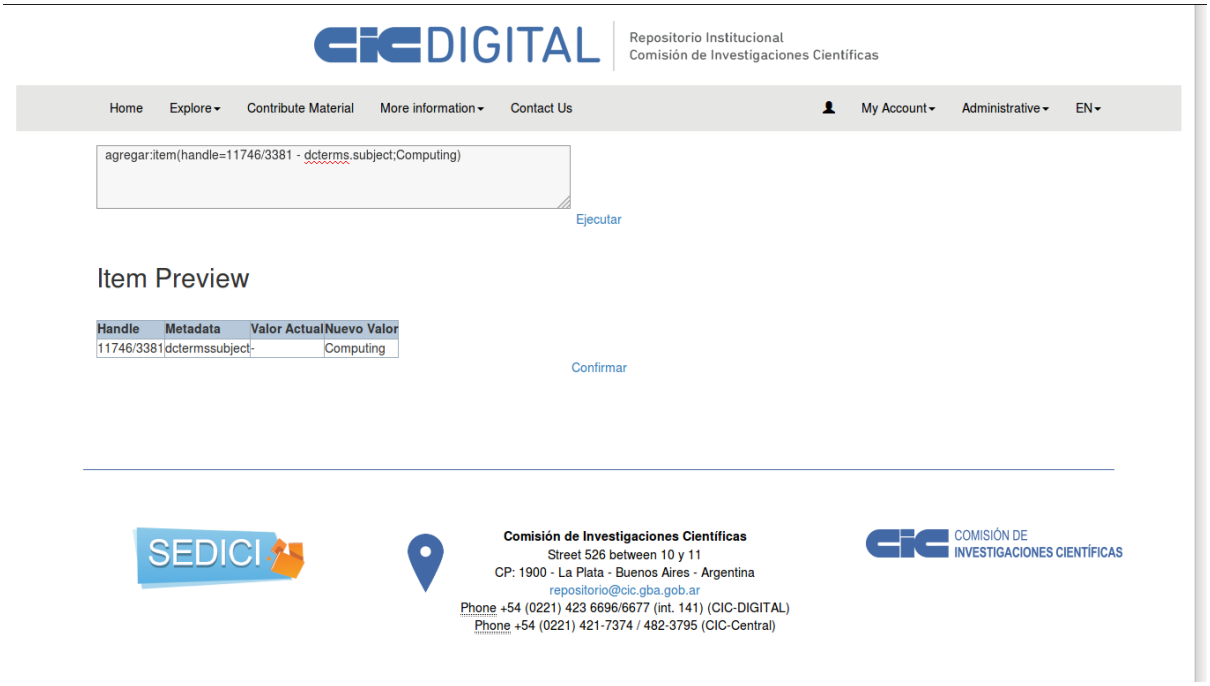


Figura 8.9: Se muestra la consulta de agregación de metadato junto con la vista anticipada correspondiente

En la figura 8.9 puede apreciarse la consulta escrita en el textarea que provee la herramienta junto con la tabla que muestra las transformaciones a realizarse, en ella puede verse que en la columna “Metadata” se indica el metadato “dcterms.subject”, en la columna “Valor actual“ se indica el valor “-” dado que la consulta es de agregación de metadatos, y en la columna “Nuevo Valor” se indica el valor “Computing”. A continuación se muestra la figura 8.10 la cual muestra el estado del Ítem luego de la confirmación de la realización de las transformaciones, en ella puede apreciarse que el Ítem tiene un nuevo valor en el metadato “dcterms.subject”.

Figura 8.10: Se muestra el estado del Ítem luego de la efectiva realización de la transformación

Como segundo ejemplo de consulta de agregación de metadatos se mostrará la ejecución de una consulta que agrega un metadato a un elemento, además se utilizará la funcionalidad provista por la herramienta llamada “Referenciar a otro metadato” la cual permitirá que el nuevo metadato que va a ser agregado tenga como valor el valor de otro metadato ya existente, dicha funcionalidad fue definida con más detalle en el capítulo 6 sección “Referenciar a otro metadato”.

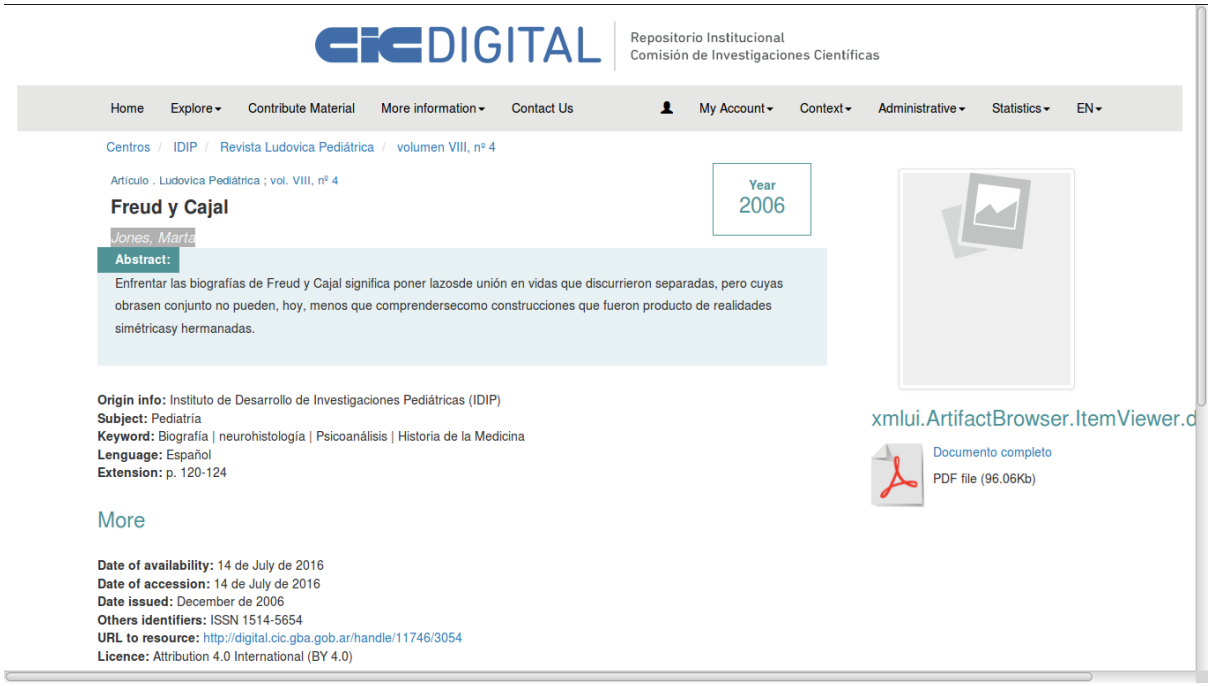


Figura 8.11: Se muestra el estado del Ítem previo a la ejecución de la consulta

Como puede verse en la figura 8.11 se muestra un Ítem cuyo autor es “Jones, Marta”, este Ítem tiene como handle 11746/3054, y se le agregará el metadato “dcterms.creator.editor” con el valor del metadato “dcterms.creator.author” (autor del Ítem). Para ello la consulta a ejecutar es la siguiente:

agregar:item(handle=11746/3054 - dcterms.creator.editor;$dcterms.creator.author)

Figura 8.12: Se muestra la vista previa de la ejecución de la consulta de transformación que permite agregar un metadato

Como puede verse en la figura 8.12 en la vista anticipada de la transformación se muestra la tabla explicada en diferentes ocasiones precedentes con la información del Ítem a transformar. En este caso particular se muestra el la columna “Metadata” el valor “dcterms.creator.editor” el cual es el metadato que va a ser agregado, en la columna “Valor actual” se ve el valor “-” el cual representa que no existe un valor actual y en la columna de “Nuevo valor” se puede ver “Jones, Marta” el cual también es el valor del metadato “dcterms.creator.author”, esto significa que la funcionalidad de *Referenciar a otro metadato* funciona correctamente.

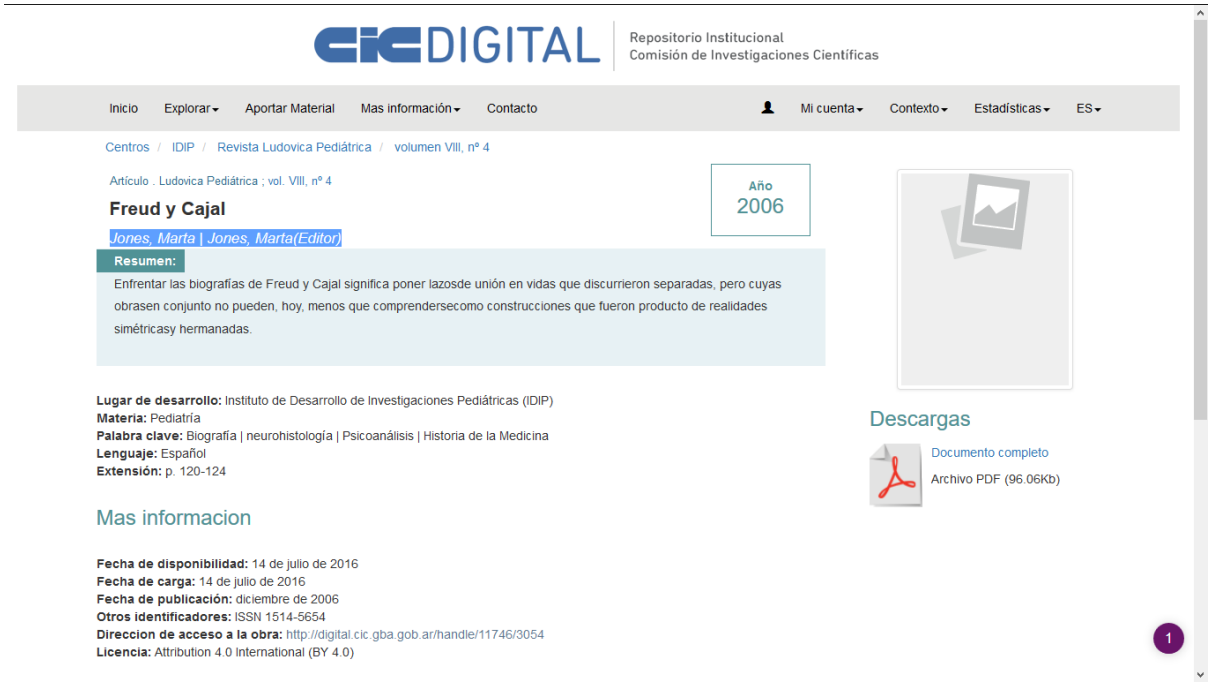


Figura 8.13: Se muestra el estado del Ítem luego de la efectiva ejecución de las modificaciones

En la figura 8.13 se puede observar el estado del Ítem en cuestión luego de la confirmación de la ejecución de la consulta, puede apreciarse que se agregó como editor a “Jones, Marta”

Eliminación de metadatos

En esta subsección se muestra la ejecución de una consulta de eliminación de metadatos, la misma se realiza de igual forma que todas las precedentes, es decir con la imagen del Ítem antes de que se efectúe la transformación, la imagen de la consulta y la vista previa y finalizar con la imagen del Ítem luego de efectuada la transformación.

Para realizar lo explicado en el párrafo precedente se comienza explicando que el Ítem elegido para realizar la eliminación del metadato es aquel con handle 11746/3381 y título “ Integrando UML y DSL en el enfoque MDA”, en esta consulta se eliminará el metadato “dcterms.subject.materia” utilizando la siguiente consulta:

eliminar:item(handle = 11746/3381 - dcterms.subject.materia)

A continuación se muestra la figura 8.14 en la cual se puede apreciar el estado del Ítem antes de que se efectúe la transformación, en este caso particular en dicha imagen se puede ver que el Ítem en cuestión posee un solo metadato “dcterms.subject.materia” el cual tiene como valor “ Ciencias Informáticas“.

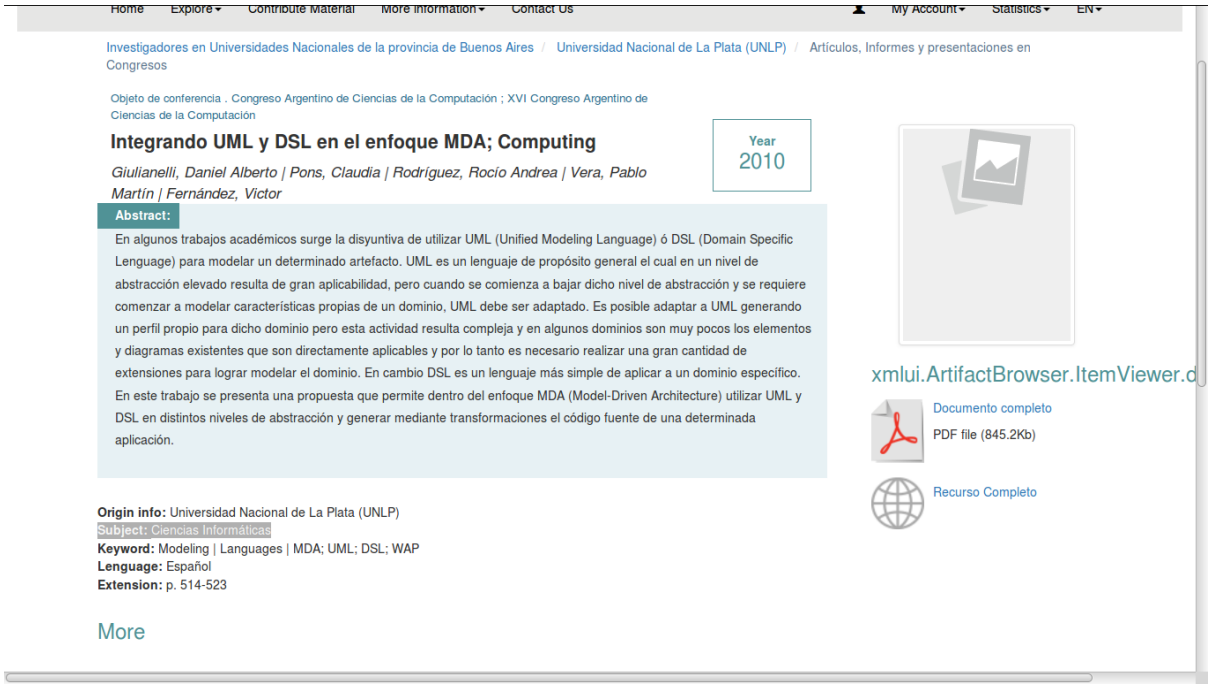


Figura 8.14: Se muestra el estado del Ítem antes de efectuada la transformación

Continuando con el ejemplo se muestra la consulta escrita en el campo que provee la herramienta junto con la vista previa que se genera a partir de la misma.

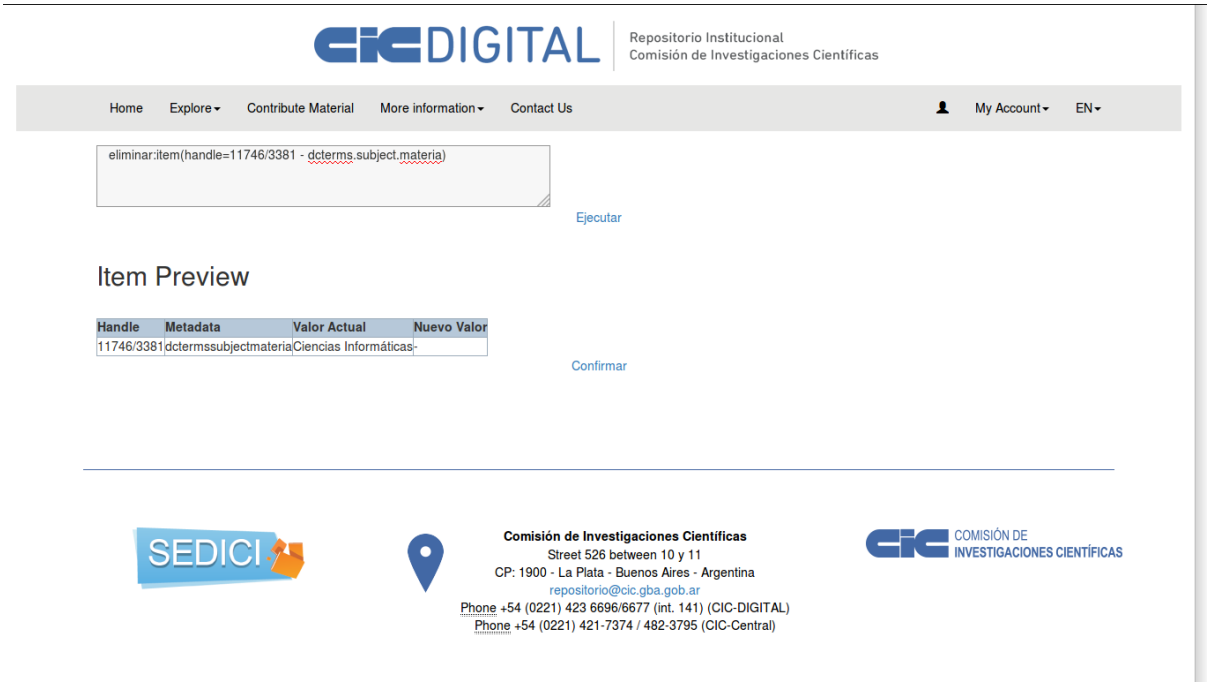


Figura 8.15: Se muestra la consulta a ejecutar junto con la vista previa generada

Como puede verse en la figura 8.15 la vista previa presenta en la columna "Metadata" el valor "dcterms.subject.materia", en la columna "Valor actual" se encuentra el valor "Ciencias Informáticas" mientras que en la columna "Nuevo valor" se encuentra el valor "-", el cual se debe a que la consulta es de eliminación y no habrá nuevo valor luego de la ejecución de la transformación. A continuación se muestra la figura 8.16 en la cual puede apreciarse el estado final del Ítem luego de la confirmación de la ejecución de la transformación, en la cual se puede apreciar que el metadato "dcterms.subject.materia" fue eliminado.

Figura 8.16: Se muestra el estado final del Ítem luego de la ejecución de la transformación

Ejemplos adicionales

Para finalizar esta sección del capítulo 8 se presentan a continuación 2 situaciones que se cree vale la pena destacarlas dado que demuestran cómo responde la herramienta ante distintas situaciones posibles. En primer lugar se demuestra la forma en la cual responde la herramienta ante una consulta sin resultados, es decir ante la ejecución de una consulta la cual no tiene efecto alguno en el repositorio, el ejemplo concreto que se provee es el de una consulta de selección la cual no tiene elementos que coincidan con las condiciones de selección propuestas, ante esta situación la herramienta le muestra al administrador un mensaje explicativo de la situación.

Figura 8.17: Se muestra el mensaje que brinda la herramienta ante una consulta sin resultados

Como ejemplo final se muestra como responde la herramienta ante una consulta de transformación la cual afecta a diversos elementos del sistema, concretamente hablando se muestra una consulta que recupera todos los elementos del repositorio cuyo metadato “dcterms.creator.author” contenga (like) la palabra “Schina” y reemplazara para cada uno de esos elementos todas las ocurrencias de la palabra “Schina” en el metadato “dcterms.creator.author” por la palabra “Shinca”. Este ejemplo trata de demostrar la forma en la cual responde la herramienta ante una consulta de transformación de múltiples elementos y a su vez brindar un posible ejemplo real el cual podría ser la modificación del nombre de un autor que haya sido cargado equivocadamente.

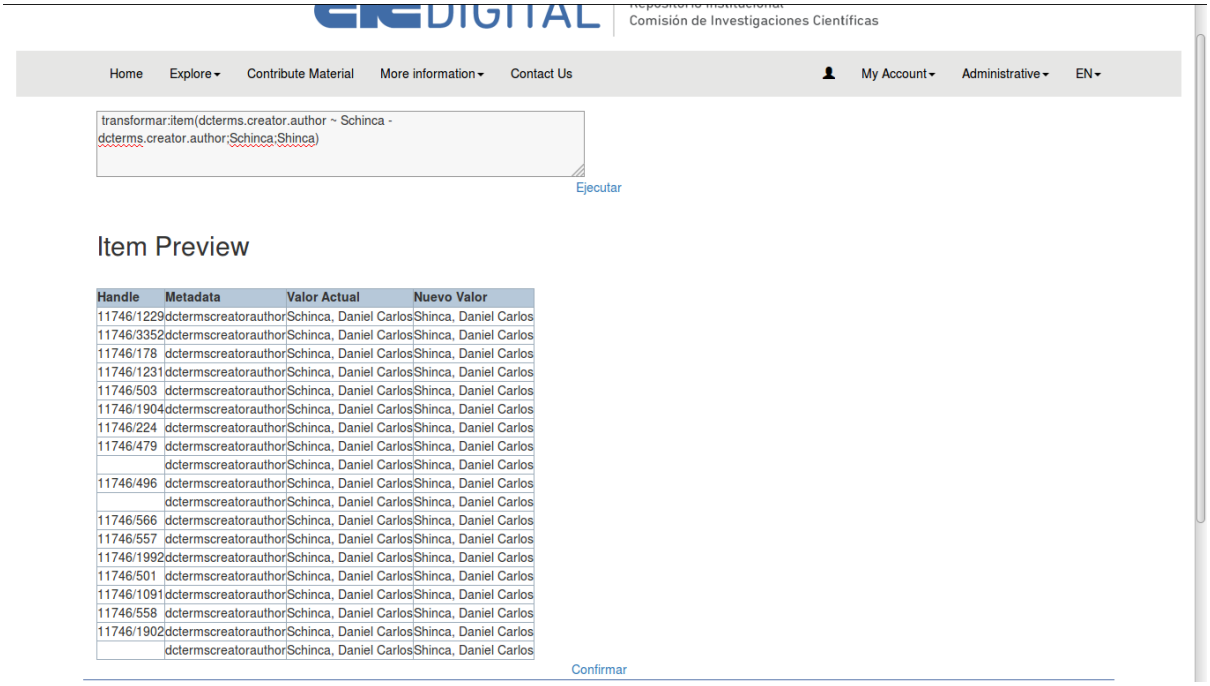


Figura 8.18: Se muestra la consulta de transformación de múltiples elementos junto con la vista previa que genera la herramienta

Conclusión

A lo largo de este trabajo se explicaron las distintas motivaciones por las cuales se realizó el mismo, también se detalló la investigación de del marco teórico junto con el análisis de los requerimientos funcionales para luego llevar al lector a través de la implementación de la herramienta y finalmente mostrar los casos concretos de uso de la herramienta en producción. Luego de realizar las acciones mencionadas y de evaluar la potencia y flexibilidad alcanzada por la herramienta desarrollada en esta tesina se detallan las conclusiones, las cuales de dividen en distintas secciones.

Implementación del lenguaje de consulta

La implementación del lenguaje de consulta fue una de las tareas más complicadas al momento de desarrollar la herramienta, dado que se debía encontrar un balance entre la potencia que debía brindar, la flexibilidad que debía presentar y la simplicidad que debía poseer. Una herramienta extremadamente potente y flexible puede llevar a un lenguaje de consultas complejo de entender y utilizar incluso para un programador, por otro lado un lenguaje poco potente y sin flexibilidad alguna podría ser simple de escribir y de entender, pero las funcionalidades que brindaría seguramente serían muy pocas y por lo tanto el lenguaje quedaría en desuso. Es por estas razones que durante el desarrollo del lenguaje de consulta que provee la herramienta desarrollada en esta tesina, se buscó que tuviera potencia y flexibilidad para cumplir los objetivos planteados pero sin dejar de lado la simplicidad, ya que esta herramienta está pensada para ser usada por administradores los cuales no necesariamente poseen conocimientos en programación.

Implementación de la herramienta

Para la implementación de la herramienta se buscó, principalmente, mantener un código simple y limpio. Para ello se dividió el código en distintos módulos (explicados en el capítulo precedente) donde cada uno cumple una funcionalidad específica, lo cual permite mantener un código más simple de entender y fácil de mantener. Asimismo para aumentar la facilidad de entendimiento del código se buscó que los nombres de los métodos, clases y variables sean expresivos, es decir que den una idea de lo que representan y a su vez dichos nombres se encuentran en inglés y respetando la forma de escritura *camelCase*. Dichas acciones tienen como objetivo simplificar futuras mejoras y modificaciones en la herramienta, así como también aumentar la posibilidad de que se pueda realizar un merge exitoso a DSpace.

Objetivos cumplidos

En el primer capítulo de esta tesina se plantearon distintos objetivos y a lo largo del desarrollo de la herramienta propuesta se fueron cumpliendo algunos y otros no pudieron llegar a realizarse, como se explicará más adelante en la sección de trabajos futuros. Para el desarrollo de esta tesina se hizo énfasis en el objetivo principal y en aquellos objetivos secundarios esenciales para el correcto funcionamiento de la herramienta, en conclusión se cumplió con el objetivo principal el cual era desarrollar una herramienta que permitiese manipular los recursos de un repositorio dado y sus metadatos asociados, a través de un lenguaje específico de uso simple y adaptado al modelo de DSpace, a su vez se cumplieron los objetivos secundarios de permitir operaciones de validación, selección y modificación sobre los elementos y sus metadatos y de Integrar la herramienta con el repositorio CIC-Digital.

Trabajo futuros

Existen diversas líneas de mejoras que se pueden realizar a la herramienta desarrollada en esta tesina, algunas de ellas fueron detectadas durante la realización de la misma, mientras que otras fueron planteadas ante el surgimiento de nuevas necesidades. A continuación se detallan las distintas líneas de mejoras previamente mencionadas con una explicación de su utilidad y necesidad.

Mejorar la interfaz gráfica

Durante el desarrollo de la herramienta se hizo foco en distintos aspectos tales como la expresividad del lenguaje, la facilidad de uso de la herramienta y el buen desarrollo de la misma, entre otros. Sin embargo hay un aspecto sobre el cual no puso demasiado énfasis: el diseño gráfico de la herramienta, esta característica en cualquier herramienta resulta de gran importancia dado que si una aplicación o herramienta no posee una buena y amigable interfaz gráfica puede llevar a una mala predisposición de los usuarios al momento de usarla o complejizar su uso. Es por estas razones que se plantea como trabajo futuro la mejora de la interfaz gráfica de la herramienta.

Testeo de la expresividad del lenguaje

Como fue mencionado anteriormente la herramienta desarrollada en esta tesina está pensada para ser usada, principalmente, por los administradores del repositorio CIC-

DIGITAL. Dichos usuarios finales no se encuentran familiarizados con la programación ni tampoco con lenguaje de consultas, es por estas razones que el lenguaje que provee la herramienta debe ser fácil de aprender, utilizar y a su vez debe ser expresivo. Con este objetivo en mente resulta necesario realizar pruebas de expresividad, de facilidad de uso y aprendizaje con los usuarios finales. Esto permitirá adaptar el lenguaje a las necesidades y facilidades de los administradores y además incentivará su uso, ya que si el lenguaje resulta complejo para ellos es probable que quede en desuso.

Realizar un Pull Request a DSpace

Queda como trabajo futuro el objetivo secundario de realizar un Pull Request (PR) al software DSpace, esto ocurre dado que se considera que antes de realizar el PR el lenguaje debe ser probado por los administradores del sistema y modificado en base a las necesidades y sugerencias que surjan. El objetivo de primero realizar el testeo de la expresividad y usabilidad del lenguaje y luego las correspondientes modificaciones es el de tener un lenguaje lo más completo posible al momento de realizar el PR, a su vez poder corroborar haber realizado dicho testeo, las correspondientes modificaciones y un nuevo testeo por parte de los usuarios finales sería un punto importante en la aceptación del PR, dado que se podría mencionar y demostrar que la herramienta fue probada por usuarios finales reales, que fue adaptada a su necesidades y capacidades y que el resultado final fue positivo.

Integrar la herramienta con otras funcionalidades de DSpace

La herramienta desarrollada en esta tesina presenta un lenguaje de consulta capaz de recuperar cualquier elemento alojado en el sistema, así como también permite realizar distintas transformaciones sobre los mismos (las cuales fueron explicadas en el capítulo 6 de esta tesina), a su vez el lenguaje que provee se implementó buscando que sea simple de aprender y utilizar para que pueda ser usado por usuarios sin conocimientos en programación. Estas características permiten que la herramienta pueda ser utilizada en distintos módulos del software DSpace, a continuación se detallan algunos ejemplos posibles.

Módulo de estadísticas

Actualmente DSpace provee un módulo de estadísticas (*DuraSpace Wiki*, 2017) el cual permite recopilar información sobre un elemento en particular del repositorio o sobre los elementos contenidos dentro de otro, por ejemplos los Ítems dentro de una colección. Con la integración de la herramienta desarrollada en esta tesina se podrían obtener estadísticas de diferentes elementos del repositorio que cumpliesen una determinada caracterísitca, por ejemplo se podrían obtener la cantidad de visitas que tuvieron todos los Ítems cuyo metadato dc.type sea igual a Artículo. Esto brindaría una gran potencia al módulo de estadísticas ya que se podría obtener información sobre los elementos del sistema de manera mucho más fácil y rápida.

Módulo de exportación

DSpace provee diferentes mecanismos de exportación de información, entre ellos OAI-PMH (*DuraSpace Wiki*, 2017) el cual es un protocolo de interoperabilidad entre repositorios, él mismo define ciertas normas que deben respetarse que permiten que

diferentes repositorios puedan fácilmente intercambiar información: el funcionamiento detallado del protocolo OAI-PMH puede encontrarse en la referencia citada, pero no será explicado en esta tesina dado que excede los límites de la misma. Una de las funcionalidades que provee el protocolo mencionado es definir qué elementos van a ser exportados, es decir si se quisiera se podría seleccionar subconjuntos de elementos en lugar de todos los elementos del repositorio. Es para potenciar dicha funcionalidad que sería útil la incorporación de la herramienta desarrollada en esta tesina, dado que con ella y utilizando su módulo de selección y condiciones de validaciones se podrían realizar subconjuntos de elementos que cuenten con características específicas con tan solo una consulta.

Módulo de búsqueda (discovery)

Una de las características principales de los repositorios basados en el software DSpace es la de permitir a los usuarios realizar búsquedas minuciosas sobre los elementos del repositorio, esto puede realizarse a través de la herramienta de búsqueda (discovery) (*DuraSpace Wiki*, 2017) que provee.

Figura 8.19: Se muestra la funcionalidad de Búsqueda y Discovery que provee el repositorio CIC-DIGITAL

Como puede verse en la figura 8.19, el repositorio CIC-DIGITAL permite realizar la búsqueda deseada escribiendo los términos de búsqueda en la barra horizontal que puede apreciarse en dicha figura, en el caso del ejemplo dado se buscó “De Guisti”, y a su vez permite realizar filtros utilizando la barra lateral, la mencionada barra lateral se conoce como discovery (*DuraSpace Wiki*. 2015) y está diseñada para permitir a los usuarios refinar su búsqueda, para lograrlo el módulo discovery (*DuraSpace Wiki*. 2015) permite agregar filtros a la búsqueda tales como:

- Nombre del autor de la obra
- Fecha de publicación

- Palabras clave de la obra
- Tipo de documento

El conjunto de estos filtros permiten a los usuarios realizar búsquedas específicas, sin embargo integrar la herramienta desarrollada en esta tesina al módulo de búsqueda permitiría aumentar todavía más la potencia de dicho módulo, ya que se podrían combinar los filtros que permite agregar discovery junto con la selección y filtros más específicos que permite agregar la herramienta. Dicha combinación permitiría a los usuarios realizar búsquedas mucho más detalladas y específicas por lo que obtendrían los elementos que desean de una manera más rápida y fácil, disminuyendo la cantidad de resultados inservibles para el usuario y disminuyendo también el tiempo de búsqueda.

Aumentar la potencia de la herramienta

Para finalizar esta tesina se describirán distintas funcionalidades que podrían ser agregadas a la herramienta las cuales le permitirían aumentar su potencia. Cada una de las 3 mejoras planteadas a continuación potenciará un tipo de consultas en específico (validación, selección, transformación).

Mejorar el módulo de selección

Como se dijo en el capítulo 6 la herramienta permite seleccionar elementos en base al handle de su elemento padre, es decir permite seleccionar Ítems en base al handle de la colección a la cual pertenecen. Pero esta funcionalidad se ve restringida por el mismo DSpace, ya que no permite seleccionar colecciones en base al handle de su comunidad padre ni seleccionar subcomunidades en base al handle de su comunidad padre. Al momento de agregar las funcionalidades mencionadas las consultas correspondientes no deberían ser diferentes a la consulta que permite seleccionar Ítems en base al handle de su padre:

seleccionar:item(handleDeColeccion)

Mantener la misma sintaxis al momento de agregar nuevas funcionalidades es fundamental para no confundir a los administradores ni complicar el uso de la herramienta. Manteniendo la misma sintaxis la herramienta adquiere más potencia sin modificar su forma de uso ni obligando a los usuarios finales a aprender un nuevo tipo de consulta ni sintaxis.

Mejorar el módulo de validación

Con el paso del tiempo y el continuo uso de la herramienta surgirán distintos requerimientos que solo podrán ser satisfechos agregando nuevas funcionalidades y características a la herramienta. A continuación se listan algunas funcionalidades que se cree van a ser de gran utilidad para los administradores y que también aportaran a la aceptación del PR de la herramienta.

Permitir manejo de todos los DSO

Como se expresó a lo largo de esta tesina, la herramienta maneja 3 tipos de DSO: Ítem, Colección y Comunidad. Estos tres DSO son los elementos básicos y esenciales en los repositorios, sin embargo en capítulo 2 de esta tesina se mostró el modelo de DSpace el cual está conformado por 6 elementos, agregando a los 3 ya mencionados: Bundle,

BitStream y BitStreamFormat. Estos últimos 3 elementos también son considerados DSO y el hecho de que la herramienta permitiese operar sobre ellos aumentaría en gran medida su potencia y alcance, dado que si esto ocurre no quedaría elemento alguno en el repositorio que no pueda ser validado, seleccionado o transformado por la herramienta.

Una posible utilidad de agregar esta funcionalidad sería poder validar si un BitStream tiene asociado un nombre y una descripción, además el poder manejar cualquier tipo de DSO permitiría poder corregir esta situación. Otro posible caso de uso para esta funcionalidad sería poder verificar si un BitStream posee un formato existente o si ningún formato fue asociado a él. Finalmente otro caso de uso sería poder evaluar si un Ítem posee por lo menos un BitStream, lo cual será explicado con más detalle en la siguiente sección

Elemento contiene a otro elemento

Como fue mencionado anteriormente las comunidades pueden contener otras Comunidades o Colecciones, las Colecciones pueden tener Ítems, estos contienen Bundles los cuales contienen BitStreams y estos a su vez tienen asociado un BitStreamFormat. En este contexto resulta sin sentido tener una Colección la cual no posea ningún Ítem, o tener un Bundle sin Bitstreams.

Es en base a lo dicho en el párrafo anterior que una mejora a la herramienta desarrollada en esta tesina sería permitir validar si un DSO contiene a otro DSO, es decir poder validar si una Comunidad contiene al menos un Ítem (por transitividad), o su un Ítem contiene al menos un BitStream. Poder realizar esta validación permitirá identificar aquellos elementos del sistema que fueron creados en algún momento y luego nunca utilizados, o también elementos que hayan sido creados equivocadamente, poder realizar esta acción con tan solo escribir una consulta resultaría de gran utilidad dado que de otra manera para validar esta situación se deberían recorrer todos los elementos del sistema manualmente.

Identificar elementos iguales

Siendo que en un repositorio existen enormes cantidades de elementos (en CIC-DIGITAL hay más de 4000) es posible que ocurra que un administrador cargue por equivocación 2 veces el mismo elemento. Detectar esta situación resulta una tarea muy dificil de realizar si se hace manualmente, dado que para realiza se debería confiar en la memoría del administrador al momento de cargar un elementos para que recuerde que ese ya fue cargado o bien comparar cada uno de los elementos del repositorio para identificar si hay alguno repetido. Claramente ambas opciones son complicadas y poco probables de que se realicen. Es por estas razones que sería de gran utilidad que la herramienta permita identificar si existen dos elementos iguales en el repositorio, para esto la herramienta debería permitir dos funcionalidades:

- Validar automáticamente todos los elementos del repositorio
- Validar si existe un elemento igual a uno dado

La primera funcionalidad evaluaría entre si todos los elementos que se encuentren en el repositorio con el objetivo de identificar si existen dos o más elementos iguales, como resulta claro probablemente sea una tarea de larga duración, la cual se podría dejar ejecutando desatendidamente para luego revisar los resultados y actuar en base a los mismos. La segunda funcionalidad compararía todos los elementos del repositorio contra un elemento en particular, el cual podría ser definido a través de su handle, esto permitiría identificar un caso concreto de elemento duplicado y aunque esta tarea también podría tomar un tiempo relativamente extenso el mismo sería menor a la funcionalidad antes

mencionada y a su vez podría ser ejecutada desatendidamente para luego verificar y evaluar los resultados.

Módulo de transformación

Actualmente el módulo de transformación permite modificar, agregar o eliminar un metadato. A su vez también permite el uso de expresiones regulares para seleccionar la parte específica del valor del metadato que va a ser modificada y también permite, en las acciones de agregar y modificar, referenciar a otro metadato para asignar su valor como el nuevo valor. Finalmente si durante el proceso de ejecución de una consulta se produce algún error que lleve a la terminación anticipada de la ejecución, todas las transformaciones realizadas se revertirán con el objetivo de no dejar inconsistencias en la base de datos.

Deshacer

Sin embargo la potencia y funcionalidades que provee la herramienta podría llevar a una equivocación de parte del administrador al momento de realizar la transformación, dado que mirando simplemente una consulta de transformación resulta difícil, incluso para los programadores, ser completamente conscientes de todas las modificaciones que se realizarán y el estado final de los elementos luego de aplicadas dichas transformaciones. Para mitigar este inconveniente la herramienta ofrece la funcionalidad de una vista anticipada, en la cual se detallan todos los elementos que van a ser transformados juntos con el metadato a transformar, el valor actual del mismo y el valor final.

Aún teniendo esta vista anticipada existe la posibilidad de una equivocación, dado que el error es una condición natural en el humano y además si la transformación involucra cientos o miles de elementos es posible que no se detecte un error al verificar los cambios que se producirán, es por estas razones que sería de gran utilidad que la herramienta permitiese volver para atrás los cambios realizados en la última consulta de transformación. Permitiendo, de esta manera, reparar errores indeseados o equivocaciones involuntarias de los administradores, asimismo resultaría un alivio al momento de ejecutar una consulta de transformación que involucra miles de elementos el saber que en caso de que se cometa un error, este pueda ser solucionado fácilmente.

Permitir el uso de expresiones regulares en la eliminación

Actualmente la herramienta permite la eliminación de metadatos a través de una simple consulta tal como:

eliminar:item(handle=11746/3381 - dcterms.subject)

Esta consulta recupera el Ítem cuyo handle es "*11746/3381*" y luego elimina su metadato "dcterms.subject", pero puede ocurrir que el elemento seleccionado posea más de una instancia de el metadato a eliminar como es el caso caso del Ítem del ejemplo, el cual posee 3 instancias del metadato "dcterms.subject", es decir que si se ejecuta la consulta ejemplificada se eliminaran las 3 instancias de dicho metadato. Es por esta razón que resultaría de gran utilidad permitir al usuario la posibilidad de agregar una condición que deba cumplir la instancia del metadato a eliminar, es decir que el usuario tenga la posibilidad de elegir si desea eliminar todas las instancias de un metadato o eliminar solo

aquellas instancias que cumplan con una determinada condición, para dar un ejemplo de lo mencionado se plantea la siguiente consulta::

eliminar:item(handle=11746/3381 - dcterms.subject;expresionRegular)

La consulta anterior eliminaría la instancia del metadato “dcterms.subject” que cumpla con la expresión regular, es decir que la consulta de eliminación de metadato funcionaria de la misma manera que las consultas de agregar y eliminar metadatos lo que conlleva a aumentar la potencia de la herramienta pero sin aumentar su complejidad, ya que al escribirse de la misma manera que las de agregación y modificación de metadatos los administradores no deberían aprender ni memorizar nuevas sintaxis.

Referencias

Arévalo, Julio Alonso (2011). Preservación digital en repositorios institucionales: Gredos. Recuperado el 22 de enero de 2017, de Eprints.rclis.org: http://eprints.rclis.org/16356/

Biblioteca Nacional de Australia. (2003). DIRECTRICES PARA LA PRESERVACIÓN DEL PATRIMONIO DIGITAL. Recuperado el 22 de enero de 2017, de http://unesdoc.unesco.org/images/0013/001300/130071s.pdf

De Giusti, Marisa Raquel & Oviedo, Néstor Fabián & Lira, Ariel Jorge & Villarreal, Gonzalo Luján. (2013). Control de integridad y calidad en repositorios DSpace. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/30524

De Giusti, Marisa Raquel. (2014). Una metodología de evaluación de repositorios digitales para asegurar la preservación en el tiempo y el acceso a los contenidos. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/43157

De Giusti, Marisa Raquel. (2015). Taller de preservación. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/52941

De Giusti, Marisa Raquel & Villarreal, Gonzalo Luján & Plangger, Lea 2015. (2015). Curso: Preservación digital de documentos. Archivos, bibliotecas y museos. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/44406

Digital.cic.gba.gob.ar. (2017). *Inicio*. Accedido el 22 de enero de 2017, disponible en: http://digital.cic.gba.gob.ar/

Dspace.org. (2017). *DSpace | DSpace is a turnkey institutional repository application.*. Recuperado el 22 de enero de 2017, disponible en: http://www.dspace.org/

Dublincore.org. (2017). *DCMI Specifications*. Accedido el 22 de enero de 2017, disponible en: http://dublincore.org/specifications/

DuraSpace Wiki. (2015). *Curation System - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Curation+System

DuraSpace Wiki. (2015). *Discovery - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Discovery

DuraSpace Wiki. (2015). *OAI - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/OAI

DuraSpace Wiki. (2015). *About Data Formats - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSPACE/About+Data+Format

DuraSpace Wiki. (2017). *Functional Overview - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Functional+Overview#FunctionalOverview-OptimizedforGoogleIndexing

DuraSpace Wiki. (2017). *Functional Overview - Model - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Functional+Overview#FunctionalOverview-DataModel

DuraSpace Wiki. (2017). *Metadata for all DSpace objects - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSPACE/Metadata+for+all+DSpace+objects

DuraSpace Wiki. (2017). *DSpace api - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSPACE/DSpace+Service+based+api

DuraSpace Wiki. (2017). *Curation System - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Curation+System

DuraSpace Wiki. (2017). *Statistics and Metrics - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/Statistics+and+Metrics

DuraSpace Wiki. (2017). *OAI- PMH - DSpace 6.x Documentation - DuraSpace Wiki*. Recuperado el 22 de enero de 2017 de Wiki.duraspace.org Sitio web: https://wiki.duraspace.org/display/DSDOC6x/OAI

Handle. (2017). *Handle Proxy*. Recuperado el 22 de enero de 2017, de Hdl.handle.net: https://hdl.handle.net/

Jcp.org. (2017). JSR-000341 Expression Language 3.0. *The Java Community Process(SM) Program - communityprocess*. Recuperado el 22 de enero de 2017, de https://jcp.org/aboutJava/communityprocess/final/jsr341/index.html

Manual, U. P. S. (1984). 4.2 Berkeley Software Distribution. *Computer Systems Research Group, Computer Science Division, University of California, Berkeley, California*.

Strodl, Stephan & Becker, Christoph & Neumayer, Robert & Rauber, Andreas (2007). How to choose a digital preservation strategy: Evaluating a preservation planning procedure. Actas de la conferencia de 2007 sobre bibliotecas digitales. Recuperado el 22 de enero de 2017 de ACM Digital Library: https://dl.acm.org/citation.cfm?doid=1255175.1255181

Terruzzi, Franco Agustín. (2015). Herramienta de validación aplicada a las tareas de gestión de calidad en un repositorio digital. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/49140

Terruzzi, Franco Agustín & Lira, Ariel Jorge & Villarreal, Gonzalo Luján & De Giusti, Marisa Raquel. (2014). Evaluación automática de preservación mediante la utilización de tareas de curación en DSpace. Recuperado el 22 de enero de 2017, de SEDICI Sitio web: http://sedici.unlp.edu.ar/handle/10915/44573

UNESCO (2003). CARTA PARA LA PRESERVACIÓN DEL PATRIMONIO DIGITAL. Recuperado el 22 de enero de 2017, de UNESCO: http://portal.unesco.org/ci/en/files/13367/10676067825Charter_es.pdf/Charter_es.pdf

Voelter, M. (2013). DSL Engineering: Designing, Implementing and Using Domain-Specific Languages: Markus Voelter: 9781481218580: Amazon.com: Books. Recuperado el 22 de enero de 2015, a partir de http://www.amazon.com/DSL-Engineering-Designing-Implementing-Domain-Specific/dp/1481218581