View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by SEDICI - Repositorio de la UNLP

UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

Titulo: Herramienta de validación aplicada a las tareas de gestión de calidad en un repositorio digital
Autores: Franco Agustín Terruzzi
Director: Dra. Marisa R. De Giusti
Codirector: -
Asesor profesional: Dr. Gonzalo L. Villarreal
Asesor profesional: Lic. Ariel J. Lira
Carrera: Licenciatura en Sistemas

Resumen

Esta tesina describe el trabajo realizado para la implementación de un aporte a los mecanismos de control de calidad existentes en un Repositorio Institucional. Se parte con el objetivo de desarrollar una herramienta dinámica de validación, que permita a los administradores del respositorio, la posiblidad de implementar ciertas tareas enmarcadas en el ámbito de la preservación de los objetos digitales.El desarrollo de esta herramienta se enmarca en el software DSpace, el sistema de gestión que utiliza el repositorio institucional de la UNLP, el SEDICI.
Para realizar el desarrollo se aborda la temática de los lenguajes específicos de dominio, y se realiza un aporte en el diseño y el análisis de un lenguaje específico de dominio (DSL) que reúna los conceptos pertenecientes al dominio de los repositorios gestionados con DSpace. Para esto, se aplican distintos conceptos pertenecientes a esta rama de investigación: metamodelos, arboles AST, sintaxis abstracta y concreta, etc.
Después, se describe la implementación desarrolllada a partir del aporte mencionado en el párrafo anterior, y informa acerca de la herramienta de desarrollo escogida para la implementación: el lenguaje de expresiones especificado en la JSR-341.
Finalmente, se documenta un caso de uso puntual para el desarrollo realizado: las tareas de curación.

Palabras Claves

Repositorios Institucionales- Control de Calidad – Preservación – DSpace - DSL – Tareas de curación – Módulo de expresiones- JAVA – JSR-341 – Lenguaje de Expresiones

Trabajos Realizados

Diseño de un lenguaje de expresiones para repositorios insitucionales, utilizando herramientas del ámbito de los lenguajes específicos de dominio.
Implementación de ese lenguaje para el caso particular del repositorio institucional de la UNLP y el software Dspace, utiilizando una herraminenta estandaradizada en la JSR-341.
Desarrollo y ejecución de casos de uso para la implementación planteada, utilizando los mecanismos de curación existentes en Dspace.

Conclusiones

La metodología utilizada garantiza un cierto nivel de abstracción sobre la implementación realizada en SEDICI.
Se obtuvo un módulo de expresiones encapsulado que puede ser referenciado desde cualquier punto de ejecución del código DSpace y que es capaz de utilizar todos los componentes del sistema subyacente.
El modelo simple y reutilizable planteado para las tareas de curación, permite la implementación de una forma relativamente sencilla, la complejidad de las validaciones que se deben realizar queda en el hecho de construir la expresión correcta en el lenguaje planteado.

Trabajos Futuros

Mejoras planteadas a la implementación del lenguaje de expresiones: soporte para identificadores persistentes y referencias a objetos genéricos; Verificación de autorizacoines de usuarios ejecutores; Implementación de operación de transformación; Soporte para concurrencia.
Mejoras para los casos de uso de la implementación: implementación de nuevas tareas de curación; Desarrollo de un módulo de exportación para Dspace utilizando expresiones; Desarrollo de un Módulo de ejecución por lotes para Dspace, que ejecute transformaciones en cadena.

Fecha de la presentación: Marzo, 2015

Dedicatoria

Para mis padres Raúl Adrián Terruzzi y Nancy Mabel Bolsón, por darme las enseñanzas más importantes de mi vida y pasarme los valores que hicieron a la construcción de la persona que ahora soy: honestidad, humildad, respeto y perseverancia.
Para mis hermanos, primos y toda mi familia, por el apoyo incondicional de siempre, la tolerancia de mis errores y la celebración de mis aciertos.

Agradecimientos

Si agradeciera a cada persona que me ayudó durante mi realización profesional, el texto de la tesina sería considerablemente más extenso. Por eso, intentaré hacer mención a aquellas personas que han hecho posible con su valioso aporte, la realización de esta Tesina

A la Universidad Nacional de La Plata, pública y gratuita, y en particular a la Facultad de Informática, por permitir que me desarrolle profesional y personalmente durante estos años y que conozca a personas muy valiosas para mí.
A la Dra. Marisa De Giusti, directora de esta Tesina, por proporcionarme las condiciones de trabajo necesarias para mi desarrollo personal, su constante confianza en mí y sus sabias correcciones de los textos de este trabajo.
A mi asesor profesional, el Lic. Ariel J. Lira, por guiarme en la realización de esta Tesina, por estar siempre dispuesto a ayudarme, mostrarme el camino correcto, por ser el ejemplo de persona y de profesional que espero ser algún día y, por sobre todo, por su infinita paciencia para conmigo, en el día a día del desarrollo de este trabajo.
A mi asesor profesional, el Dr. Gonzalo L. Villarreal por prestarse en cualquier momento a colaborar con lo que fuese necesario, asistirme en todos los aspectos correspondientes a la investigación y la escritura; y mostrarme el rumbo profesional que necesitaba para mi vida.
A mi amigos y compañeros del SEDICI, Agustín Marisi, Bruno Percivale y Lucas Folegotto por estar siempre presentes en los momentos difíciles de esta etapa y por colaborar arduamente con la corrección y las imágenes de este trabajo.
A Guadalupe Giusio, la persona más importante de mi vida, a quien amo y quien ha estado presente en cada momento significativo para mí, dándome su apoyo incondicional, acompañándome y alentándome en todo momento.
A mis padres y mis hermanos, por todo su apoyo en la realización de mi carrera. Y al resto de mi familia, por la compañía de siempre.

"Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer"

Alan Mathison Turing (23/06/1912- 7/06/1954)

Índice

Introducción

El trabajo en un repositorio institucional propone nuevas problemáticas a diario y requiere soluciones pragmáticas, pero que no se acoten al dominio de un único repositorio. La preservación y la difusión de contenidos no son sólo objetivos generales, sino que se encuentran en objetivos más específicos, surgidos del día a día.

Esta tesina se enfoca en ciertas necesidades diarias que surgen en el ámbito de la preservación de contenidos en el Servicio de Difusión de la Creación Intelectual, el repositorio institucional de la Universidad Nacional de La Plata. Estas necesidades responden a los controles que se deben realizar sobre los contenidos alojados, en el marco del plan de preservación y las directivas de la política del repositorio.

Por todo lo anterior, en el Capítulo 1 se localiza el problema, se plantean las necesidades y la motivación que lo origina y los objetivos de este trabajo, que buscan solucionarlo.

Después, en el Capítulo 2, se da una breve explicación sobre el ámbito de los repositorios institucionales en el mundo, en Argentina y en la Universidad de La Plata, seguido de una breve contextualización acerca de la preservación digital, para definir el alcance y conocer el contexto de este trabajo.

Esto se especializa al SEDICI en el Capítulo 3, donde se define el software que se utiliza para gestionar el repositorio, se lo caracteriza y se brindan datos específicos de su implementación, importantes para la solución que se planteará después. Además, se definen las tareas de curación, mecanismos de automatización muy importantes en el software e implementados en esta tesina, como casos de uso de la solución planteada.

En el Capítulo 4 se comienza a definir una solución para el problema planteado y en el entorno descrito en los capítulos anteriores. Además, se brinda un estado del arte sobre el dominio en el que la solución fue realizada y la forma en la cual se la trabajó: el desarrollo de lenguajes específicos de dominio.

Con el objetivo de ir desde lo más general hacia los casos más específicos, en el capítulo 5 se caracteriza la solución planteada, utilizando ciertos mecanismos de lenguajes específicos de dominio, para que la especificación de la solución y su implementación, sean en sí mismos, aportes de este trabajo, pudiéndose, por ejemplo, en el futuro pensar en otra posible implementación para la especificación aquí brindada.

Lo de este último capítulo se concreta en el capítulo 6, donde se define una implementación para la especificación anterior, a partir de la descripción de cada etapa de esta implementación: desde el diseño, hasta la configuración.

Finalmente, en el capítulo 7, se da un caso de ejemplo de la implementación planteada, particularmente enfocado en las tareas de curación y su entorno de ejecución, mencionadas en el capítulo 3.

Como conclusión, el capítulo 8 analiza la especificación de la solución como lenguaje, la implementación de dicho lenguaje y la utilización ejemplo de esa implementación, a partir de las tareas de curación antes mencionadas. Además, se muestra la continuidad del proyecto con una serie de trabajos futuros y mejoras planteados para la solución descrita en los capítulos anteriores.

Capítulo 1 | Motivación y Objetivos

Motivación

El desarrollo de los repositorios institucionales, el crecimiento de sus contenidos y el reconocimiento de que la actividad institucional se canaliza, cada vez más, en soporte digital, genera que los repositorios a acompañar su desarrollo con actividades destinadas a la preservación y a la difusión[De Giusti, Lira, Villarreal, & Texier, 2012]. Como se dirá en capítulos posteriores, la difusión se realiza mediante una serie de acciones cuyo objetivo es lograr que los elementos del repositorio alcancen a la comunidad; la preservación se refiere a la persistencia en el tiempo de tales elementos, a fin de asegurar su acceso y usabilidad a largo plazo. Ambas actividades están influenciadas por los aspectos políticos, económicos y tecnológicos que rigen un repositorio.

Desde su creación en el año 2003, el Servicio de Difusión de la Creación Intelectual, el repositorio institucional de la Universidad Nacional de La Plata, ha evolucionado tanto en las políticas como las tecnologías que utiliza. A continuación, se nombran las caracterísitcas más importantes de estos dos aspectos.

Políticas

Desde el punto de vista de las políticas, en el repositorio se ha intentado hacer énfasis no sólo en la centralización y visualización de la producción científica de la UNLP, sino también en el establecimiento de una serie de requerimientos o recomendaciones, para asegurar la preservación en el tiempo de cada uno de esos elementos.

En este contexto, a lo largo del tiempo se ha observado una evolución tanto en el desarrollo de estas recomendaciones, como en distintos intentos por formalizarlas en una política que englobe a todos los servicios del repositorio y centralice los aspectos a tener en cuenta en el momento de preservar cualquier tipo de elemento.

Tecnologías

Desde el momento en el cual se pensó en establecer el repositorio, se planteó siempre la utilización de nuevas y mejores tecnologías, ya sea en el empleo de hardware novedoso o en la incorporación de nuevas herramientas de software que mejoren las prestaciones de los distintos servicios.

En la actualidad, el repositorio reúne más de 40000 obras que provienen de toda la UNLP y con tipologías muy variadas: desde artículos de revistas y tesis doctorales, hasta archivos de audio de entrevistas y algunos videos.

Todo este crecimiento sostenido y la evolución contínua a lo largo de los años, generan mucha variabilidad en las condiciones del repositorio, por lo que los controles y evaluaciones para asegurar la preservación y la difusión de los miles de items demandan grandes esfuerzos tanto en el personal como en los mecanismos que se pueden pensar para estos.

Marco de trabajo

El SEDICI trabaja en el establecimiento de un plan de preservación[De Giusti, 2014] que engloba aspectos políticos, económicos y técnicos. Una parte de ese plan de preservación en sus aspectos tecnológicos abarca la definición de una serie de validaciones a realizar sobre elementos del repositorio, con el fin de asegurar su preservación en el tiempo. Estas validaciones se discuten y se construyen, no como una solución definitiva, sino como un

enfoque iterativo. Se basan en normas generales, e incluso estándares como: el Modelo de Referencia OAIS[(OAIS), CCSDS Magenta Book. Issue 1.] o el diccionario de datos PREMIS[PREMIS Data Dictionary for Preservation Metada], y definen un modelo de reglas con un enfoque propio, que responde a las necesidades del SEDICI.

Dentro de las validaciones también se consideran las modificaciones que se podrían proponer sobre los elementos en donde se encontraron irregularidades. Además, el cuerpo de dichas modificaciones suele ser propenso a cambios surgidos de la propia experiencia o de los distintos estándares que buscan una correcta conformación de los datos.

Por lo dicho en los párrafos anteriores, surgió la necesidad de la definición de una herramienta que permita realizar validaciones y actualizaciones sobre los datos y que a la vez sea extensible y escalable.

En experiencias previas, se detectó que el principal problema en la utilización de mecanismos automáticos era causado por los complejos mecanismos de configuración: que requerían tiempos extensos y conocimientos en programación. En consecuencia, se busca que la herramienta aquí planteada tenga una interfaz de configuración lo más simple y amigable al usuario posible.

Trabajos anteriores

El tiempo de trabajo que lleva el equipo de SEDICI en los mecanismos de preservación, le ha permitido proponer, desarrollar e implementar distintos tipos de soluciones al problema planteado. Estas soluciones concretas, pueden verse a través de distintos trabajos presentados por el equipo.

En “Las actividades y el planeamiento de la preservación en un repositorio institucional”[De Giusti et al., 2012], Marisa de Giusti presenta una propuesta de naturaleza técnico-práctica con el objetivo de proponer un plan inicial de desarrollo de una estrategia de preservación de contenidos de un repositorio institucional. En la misma ya se se hace hincapié en los metadatos que soportan y documentan el proceso de preservación digital.

La propuesta y la estrategia de preservación se trabajan y se asientan con el tiempo; y más tarde en “Control de integridad y calidad en repositorios DSpace”[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[...]”* y comienzan a plantear las bases para establecer estos mecanismos a partir de la extensión de módulos de Dspace, particularmente el módulo de curación (ver capítulo 3). Además, definen una serie de posibles tareas de curation que contribuyan al control de calidad en un repositorio, incluso nombrando las tareas asociadas a los metadatos de preservación definidos en la propuesta previa. Todo esto, evidencia:

- La necesidad de contar con tareas de curación configurables tanto en el conjunto de metadatos, que serán considerados para la evaluación, como en el tipo de recurso, que será esperado como resultado en cada caso;
- La necesidad de poseer tareas programadas que se ejecuten periódicamente
- La necesidad de una tarea que pueda aplicar validaciones y generar reportes sobre los items y metadatos que no superen satisfactoriamente la ejecución de sus validadores

- La necesidad de automatizar la generación de algunos metadatos de preservación
- La necesidad de permitir definir expresiones lógicas simples (de verdadero o falso) para seleccionar o descartar un determinado elemento para su inclusión (o no) en una tarea de curación

Este trabajo continuó avanzando y se lograron definir ciertas actividades de preservación digital, que se realizan periódicamente y muchas ya se toman como políticas, dentro del flujo de trabajo del repositorio. Estas se lograron listar en “Preservación digital: un experimento con SEDICI-DSpace”[De Giusti, Lira, Villarreal, Terruzzi, & Adorno, 2014)] y en “Las actividades de preservación en un repositorio digital destinadas a dar acceso a lo largo del tiempo a sus contenidos” [De Giusti, 2014a], ambas presentaciones publicadas por muchos miembros del equipo actual del SEDICI.

Finalmente, se avanzó en el intento de automatizar y resolver las necesidad planteadas anteriormente. Para esto, se especificaron mecanismos de validación automatizados y configurables, para realizar diversos controles de calidad; que quedaron definidos y registrados en el trabajo titulado “Evaluación automática de preservación mediante la utilización de tareas curación en Dspace”[Terruzzi, Lira, Villarreal, & De Giusti, 2014] En el mismo, se menciona que la posibilidad de generar reglas para control de calidad y preservación es solo una etapa más que marca el crecimiento del control de la preservación en el repositorio. El siguiente paso sería atender la necesidad de una herramienta que permita a cualquier administrador del repositorio definir y ejecutar estas reglas de validación sin la necesidad de contar con conocimientos técnicos avanzados sobre la herramienta. De esta necesidad y de las posibles soluciones para la misma, surge la motivación de este trabajo.

Planteo de la necesidad

Con los distintos avances logrados en los mecanismos de control de calidad y, particularmente, en los orientados a la preservación de los recursos en el repositorio; se comenzó a pensar en distintos modos de uso para estos mecanismos. Si estos procedimientos tenían el objetivo de automatizar los controles de calidad, era necesario que puedan ser llevados a cabo por distintos tipos de administradores en un repositorio. No obstante, la configuración requerida para que funcionen correctamente los mecanismos, tal y como estaban implementados hasta el momento, se debía hacer a través de archivos de configuración de Dspace y de forma manual. Es decir, a partir de varias ejecuciones de estos mecanismos, se pudo notar la necesidad de un medio de configuración más ágil, por medio del cual los administradores puedan personalizar *qué* y *cómo* evaluar durante la ejecución de un mecanismo de control de calidad y cómo arreglar aquello que no ha pasado estas validaciones.

Por todo lo anterior, se plantea las necesidades que han surgido a partir del trabajo constante en el repositorio y que son el motivo de este trabajo:

- Poseer una herramienta de control de calidad configurable, donde su configuración funcione mediante alguna interfaz o proceso, que no requiera de conocimientos en programación para emplearse.
- Que la herramienta configurable sea lo suficientemente ortogonal para emplearse en los distintos módulos del repositorio, desde donde se realizan los controles de calidad (se verán en la sección siguiente, "*Casos de uso*").
- Que esta herramienta entienda y admita distintos tipos de validaciones, sujetas a la variabilidad de las reglas que se precisa evaluar y de los objetos sobre los que se van a aplicar dichas reglas.
- Que esta herramienta sea de implementación incremental y con suficiente adaptabilidad para adecuarse a los cambios tecnológicos que afectan a un repositorio.

Casos de uso

A partir de una puesta en común con los distintos tipos de usuarios del repositorio de SEDICI y de la experiencia del equipo de desarrollo en la problemática de repositorios, surgieron como casos concretos de los problemas y necesidades que se intentan solucionar con este trabajo los siguientes casos de uso posibles:

Mecanismo de consulta

Se debe permitir que un administrador realice consultas sobre el estado los objetos del repositorio desde algún tipo de interfaz de consulta. La idea es que la herramienta de control de calidad se pueda activar desde un panel de control y que permita generar reportes con los resultados de las validaciones realizadas.
En este caso, un usuario con conocimientos técnicos y con los privilegios adecuados, debería poder al menos:

- Mostrar todos los items, comunidades y/o colecciones de un repositorio
- Listar los ítems, comunidades y/o colecciones que cumplan con una condición determinada
- Listar los metadatos de un item, comunidad o colección
- Consultar si los metadatos de algún item, comunidad o colección cumplen con una condición determinada

Tareas de curación

Durante la ejecución de ciertas tareas de curación que requieren realizar distintas operaciones sobre varios de los datos de los objetos sobre los que operan, surgió la necesidad de poseer una herramienta para facilitar estas operaciones.[Terruzzi et al., 2014]

Se analizó que la herramienta debía, al menos, permitir que en el contexto de una tarea de curación, aplicada sobre uno o varios elementos del repositorio, se pudiera:

- Analizar si los metadatos del elemento tratado cumplían con una determinada condición,
- Analizar si los metadatos del elemento tratado eran los mismos que los de algún otro elemento similar,
- Listar todos los elementos que cumplían con la misma condición que cumplía el elemento analizado, y

- Definir dinámicamente la validación a comprobar por la tarea de curación. Por ejemplo, los metadatos que deben existir en el elemento analizado.

Modificación en masa de metadatos

También apareció la necesidad de que la herramienta permita la modificación de ciertos metadatos de un conjunto determinado de objetos, formado por aquellos que cumplan con alguna condición especificada y evaluada.

La idea era poseer un mecanismo que permita una modificación en masa de los metadatos de ciertos elementos del repositorio. Con el objetivo de reemplazar los metadatos erróneos, inexistentes o de formato incorrecto. Por esto, mínimamente se debe poder:

- Listar los elementos con metadatos inválidos o inexistentes,
- Actualizar un metadato determinado de un conjunto de elementos, y
- Añadir un metadato a un conjunto de elementos que no lo posee aunque debiera.

Mejorar la configuración del repositorio

Se podrían definir ciertas configuraciones para el repositorio de una manera más amigable o que requiera de menos tiempo. La idea sería que se pueda utilizar una herramienta para indicar con más simplicidad los objetos sobre los que se van a aplicar las configuraciones. Para esto, la herramienta debería permitir:

- Referenciar los elementos a los que se apunta con la configuración, por ejemplo utilizando alguna condición que deben cumplir, y
- Modificar ciertos elementos para que se ajusten a una configuración nueva.

Mejorar el módulo de búsqueda

La posibilidad de que ciertos usuarios avanzados puedan realizar mejores búsquedas en el módulo de búsqueda[(DuraSpace, Discovery, s. f.)] del repositorio a partir de la posibilidad de crear consultas más eficientes y correctas. Se podría utilizar la herramienta, para mejorar la creación de las consultas al módulo de búsqueda. Para hacer esto, la herramienta debía dar soporte para:

- Evaluar una condición sobre los elementos de un repositorio,
- Recuperar los elementos que cumplen con una condición,
- Acceder a los valores de distintos metadatos de un elemento, y
- Comparar elementos del mismo tipo.

Planteo del problema principal

Con el análisis de los casos de uso, se observó que una gran parte de los mismos podría ser resuelta si se tuviera un mecanismo dinámico y configurable para validar cierta información en un repositorio, para generar una serie de reportes sobre los elementos que verifiquen esta validación y para ejecutar las transformaciones pertinentes que requieran las distintas validaciones realizadas.

Además, la experiencia con repositorios digitales y el análisis de expertos en el dominio mostraron(puede verse en [De Giusti, 2014b]*, [Terruzzi et al., 2014]*) que era necesario que la herramienta a proponer pueda ser gestionada por un administrador con conocimientos avanzados en el dominio de los repositorios, y con conocimientos en programación menores.

En resumen, se plantea una herramienta para realizar controles de calidad relacionados a preservación que sea lo suficientemente amplia como para adaptarse a varios casos de uso distintos y que pueda ser utilizada por un usuario que no necesariamente deba poseer conocimientos en la programación del software de repositorios. En este contexto, el planteo de

una solución a este problema se formalizará en los objetivos generales y específicos que se describen a continuación.

Objetivos

Objetivo General

El planteo del problema descrito en la sección anterior ha dado lugar al objetivo general de este trabajo. Como el equipo del SEDICI ha encontrado recurrente al problema, el objetivo general es proponer una solución a la falta de mecanismos de control de calidad dinámicos en un repositorio institucional. En conjunto con este objetivo general, se ha buscado analizar el dominio del problema y describir algunos contextos que pueden servir para justificar las decisiones tomadas para el desarrollo de una solución.

Objetivos Específicos

Como resultado de una visión analítica de los objetivos generales, se obtuvo una lista de objetivos específicos más concretos, en los que se centra el desarrollo del trabajo.
Los objetivos pensados fueron:

- Estudiar el contexto de los Repositorios Digitales y su problemática con respecto a la preservación y al control de calidad.
- Diseñar una herramienta de validación dinámica, que permita realizar una serie de validaciones y transformaciones sobre los datos de un repositorio digital.
- Estudiar distintas alternativas de implementación que permitan lograr una herramienta lo suficientemente general como para que se pueda implementar en distintos contextos y con diferentes mecanismos.
- Construir y especificar una herramienta para facilitar ciertas tareas recurrentes en la preservación y la curaduría.
- Dar una posible solución al planteo del problema de este trabajo, a través de esta herramienta.
- Plantear, a partir de esta herramienta, un mecanismo de control de calidad y preservación continua, mediante la automatización de tareas específicas.
- Mostrar al menos un caso de uso concreto, que se haya solucionado utilizando la herramienta desarrollada.

Capítulo 2 | Repositorios y Preservación

Concepto de Repositorio

Se consideran repositorios digitales a aquellas colecciones digitales de cierto tipo de producción, ya sea temática, institucional, artística, etcétera, en las que se permite el almacenamiento, la búsqueda y la recuperación de información. Un repositorio digital contiene mecanismos para importar, identificar, almacenar, preservar, recuperar y exportar un conjunto de objetos digitales, normalmente desde un portal web. Esos objetos son descritos mediante metadatos que facilitan su recuperación. A su vez, los repositorios digitales son abiertos e interactivos, pues cumplen con protocolos internacionales que permiten la interoperabilidad entre ellos.[Productiva, SNRD s. f.]

Repositorios Institucionales

Se puede definir a un repositorio institucional como un conjunto de servicios centralizados, creados para organizar, gestionar, preservar y ofrecer acceso a una comunidad designada a la producción científica, académica, administrativa o de cualquier otra naturaleza, en soporte digital, generada por los miembros de una institución. En la actualidad existe una fuerte tendencia hacia la generación de repositorios institucionales abiertos, en los que la producción allí almacenada se expone bajo licencias abiertas (por ejemplo la familia de licencias Creative Commons) que promueven su uso y maximizan el impacto y la difusión de los documentos.

De estas definiciones, se pueden resumir las principales características de un RI:

- Su naturaleza institucional, entendiendo por institución a una organización educativa y/o de investigación, cuyo punto de partida fueron las universidades;
- Su carácter científico, acumulativo y perpetuo
- Su carácter abierto e interoperable con otros sistemas

A estas características se debe añadir una propia del conjunto: la diversidad. Una cualidad distintiva entre los repositorios institucionales es, justamente, que no existen dos repositorios idénticos.[De Giusti, 2014b]

A partir del análisis de las características antes mencionadas, resulta lógico creer que cada RI de una determinada institución intentará agrupar los materiales que representen la producción de dicho establecimiento. Como consecuencia de esto, un RI puede ser visto como una colección de documentos y objetos, por lo general de tipos y formatos variados.

El personal asociado de alguna forma al RI debe tener herramientas y métodos para depositar el material a su cargo o de su autoría (textos, conjuntos de datos, archivos de sonido, imágenes, etc), y es responsabilidad del RI (de acuerdo a las políticas que se implementen en la institución) cerciorarse de que el formato de dicho material sea el adecuado para preservar y difundir.

Actualidad de los Repositorios

La tendencia mundial de exponer la producción de una comunidad o institución académica a través de repositorios de acceso abierto, ha desencadenado en la creación de repositorios institucionales en todas partes del mundo. En la actualidad, se pueden encontrar más de 2700 repositorios en todo el planeta, que nuclean millones de objetos digitales. (Ver *figura 2.1*).

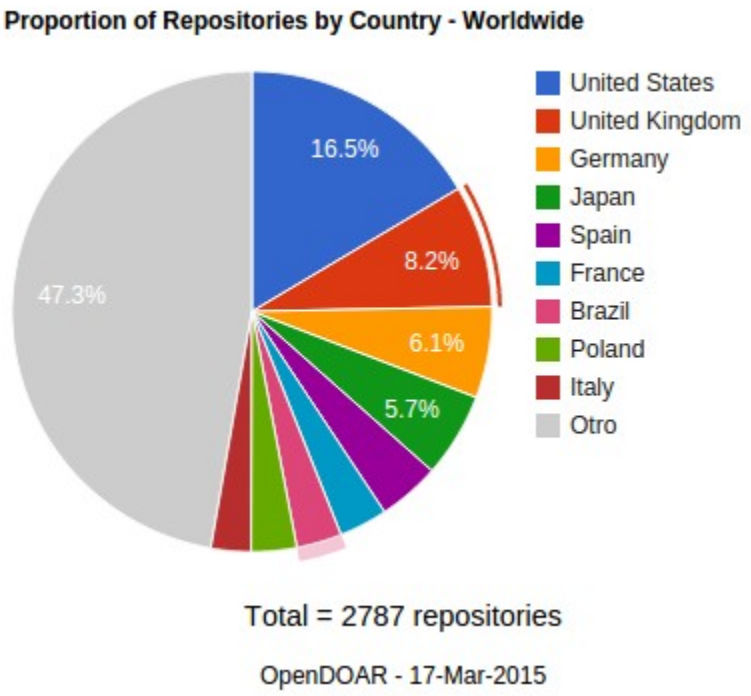

Figura 2.1: Distribución actual de los repositorios en el mundo según OpenDoar(http://www.opendoar.org/)

Entre los más importantes del mundo [Consejo Superior de Investigaciones Científicas, s. f.], se pueden mencionar:

- *Arxiv (http://arxiv.org/): Definido por su creador, desde 1991, como "un sistema de distribución automática de artículos de investigación, sin las operaciones editoriales asociadas a la revisión",* se lo suele referenciar como el primer repositorio institucional de la historia. En la actualidad, *contiene más de 1.000.000 de documentos científicos en áreas de física, matemática, ciencias de la computación, biología cuantitativa, finanza cuantitativa y estadísticas.*
- *PubMed (http://europepmc.org/):* un archivo digital libre de más de 2300 revistas de biomedicina y ciencias de la salud existentes en los Institutos Nacionales de Salud de los Estados Unidos. Este repositorio, considerado en la actualidad como el número 1 del mundo, es desarrollado y administrado por NIH's National Center for Biotechnology Information (NCBI) de la National Library of Medicine (NLM)[Information, s. f.]
- *Social Science Research Network (http://ssrn.com/en/):* ofrece una base de datos de más de 532 mil documentos científicos en ciencias sociales cubriendo más de 400 áreas, de los cuales 492 mil se pueden acceder a texto completo online proveniente de investigadores de 70 países.
- *Repec* (http://repec.org/): es un esfuerzo colaborativo para mejorar la difusión de materiales en ciencias económicas. La base de datos ofrece más de 1.7 millones de registros de más de 2000 revistas distintas.
- *NTRS(https://ntrs.nasa.gov/):* Provee acceso a unas 500 mil citas bibliográficas de documentación aeroespacial, 90 mil documentos online a texto completo y más de 100 mil imágenes y videos.

Repositorios en Argentina

En Argentina existen varios repositorios institucionales que brindan acceso a revistas científico-técnicas, tesis, informes de investigación, presentaciones a congresos y demás documentación científica de producción nacional y de acceso abierto que pueden consultarse desde cualquier punto con conexión a Internet.

La única condición, para la reproducción y distribución de las obras, es la obligación de otorgar a los autores el control sobre la integridad de su trabajo y el derecho a ser adecuadamente reconocidos y citados.

En este contexto, en el año 2011 se creó el Sistema Nacional de Repositorios Institucionales (SNRD), con el objetivo de impulsar, gestionar y coordinar una red interoperable de repositorios distribuidos físicamente, creados y gestionados por instituciones o grupos de instituciones a nivel nacional para aumentar la visibilidad e impacto de la producción científica y tecnológica del país.[Productiva, s. f.-b]

Actualmente, el SNRD se encuentra en pleno crecimiento, principalmente debido a los beneficios que presenta:

- Integración a LA Referencia: Red Federada de Repositorios Institucionales de Publicaciones Científicas resultante del Proyecto BID ATN/OC-12013-RG “Estrategia Regional y Marco de Interoperabilidad y Gestión para una Red Federada Latinoamericana de Repositorios Institucionales de Documentación Científica"[«LA Referencia |», 2012],
- Articulación con las políticas nacionales de acceso abierto que se generan desde el Ministerio de Ciencia, Tecnología e Innovación Productiva
- Incremento de la visibilidad de la producción científico-tecnológica generada por las instituciones y organismos
- Posibilidad de acceder a las líneas de financiamiento del SNRD.[Productiva, s. f.-b]

Como consecuencia de esto, los últimos números del SNRD expresan que en la actualidad:[Productiva, s. f.-d]

- 25 centros se encuentran adheridos con Resolución de la Secretaría de Articulación Científico Tecnológica.
- 2 repositorios están adheridos y a la espera de la resolución de la Secretaría
- Entre los repositorios adheridos más importantes se pueden mencionar ARGOS (UNAM)(http://argos.fhycs.unam.edu.ar/) Nülan (UNMdP-FCES) (http://nulan.mdp.edu.ar/), Memoria Académica (UNLP-FAHCE) (http://www.memoria.fahce.unlp.edu.ar/), Biblioteca Digital (UBA-FCEN) (http://digital.bl.fcen.uba.ar/gsdl-282/cgi-bin/library.cgi), el Repositorio Digital Institucional "José María Rosa" (UNLA)(http://www.repositoriojmr.unla.edu.ar/), la Biblioteca Virtual (UNL)(ref:http://bibliotecavirtual.unl.edu.ar/), etc [Productiva, s. f.-c].

Se debe mencionar que el Repositorio Institucional de la UNLP, el SEDICI, del que se dará un poco más de información en la sección siguiente, también es parte de esta red argentina de repositorios.

Caso SEDICI

El Servicio de difusión de la creación intelectual se creó en el año 2003[De Giusti, Sobrado, Lira, Vila, & Villarreal, 2008] con el objetivo prioritario de socializar el conocimiento generado en las diferentes áreas académicas de la Universidad de La Plata, con el fin de devolver a la comunidad los esfuerzos destinados a la Universidad Pública. Este objetivo engloba otros más puntuales, entre los que se pueden listar:

- Crear en la UNLP un conocimiento capaz de posibilitar la realización de un servicio de tesis digitales
- Incorporar a la UNLP al conjunto de Universidades que presentan su creación intelectual en forma abierta al mundo
- Hacer públicas estas creaciones en la comunidad local e internacional
- Generar un vínculo a nivel nacional e internacional entre quienes aportan sus creaciones y la comunidad local e internacional que las accede
- Crear una cultura local sobre el uso de bibliotecas digitales
- Facilitar un medio de utilidad a la hora de nuevas producciones para proveer ideas, antecedentes y bibliografía de modo de propender al mejoramiento de las nuevas realizaciones
- Fomentar una cultura de compartición de creaciones en un espacio común a todas las disciplinas
- Finalmente incorporar a la Universidad Nacional de La Plata a otras redes de recursos digitales ya existentes, en nuestro caso al Union NDLTD Catalog [Dissertations, s. f.]

Desde su creación, SEDICI ha afrontado diversas dificultades que han influido directa o indirectamente en su desarrollo y crecimiento [(De Giusti, 2010)]. En la actualidad cuenta con una base de datos documental que supera los 40000 recursos académicos propios (de la UNLP) expuestos bajo las políticas del Acceso Abierto. Esto convierte a SEDICI en uno de los principales exponentes en su tipo, tanto a nivel nacional como regional (América Latina)[«Latin America | Ranking Web of Repositories», s. f.].

En este contexto, la elección del software ha sido una decisión crucial durante el ciclo de vida del repositorio. En una primer etapa, SEDICI, como un portal de acceso libre, estuvo soportado por un desarrollo de software propio en PHP, MySQL y Java, llamado Celsius DL [(De Giusti, Marmonti, Sobrado, Vila, & Villarreal, 2005)], que permitía el depósito y búsqueda de recursos (objetos físicos o digitales), por ejemplo: artículos de publicaciones periódicas, preprints, tesinas de grado y tesis de postgrado, producciones multimediales, libros electrónicos, autores, revistas, eventos, comunidades, colecciones, entre otros.

Más tarde, a finales del 2011, se estudió la posibilidad de migrar a una plataforma que estuviera a la par de las nuevas tecnologías aplicadas al dominio y que fuera más ameno para el usuario y para la gestión de recursos por parte del personal. Este análisis, tenía el objetivo de comparar diferentes plataformas de software: DSpace, EPrints, FEDORA y Greenstone, dando como resultado, la elección del software DSpace como el que mejor se ajustaba a los requerimientos del repositorio.[(De Giusti et al., 2011)]

El proceso de migración de Celsius DL a DSpace finalizó en el 2012 (DSpace-SEDICI), esto permitió a SEDICI contar con funcionalidades como: búsquedas y faceting, proveedor de datos y de servicios de acuerdo con el protocolo OAI-PMH[Lagoze, Van de Sompel, Nelson, & Warner, 2015], autenticación de usuarios, facilidad en la indexación por parte de los

buscadores web, etc. Actualmente, SEDICI se encuentra en plena migración desde la version 1.8 a la 5 de Dspace, lo que muestra una constante adaptación del servicio a las tecnologías que han surgido a lo largo de los años. En el Capítulo 3 de este trabajo se describen algunas características del software Dspace, que brinda el marco de trabajo y contexto de desarrollo de esta tesina.

Cabe destacar que a partir de aquí, en este trabajo se hará referencia a SEDICI y al repositorio Institucional de la UNLP como sinónimos, especialmente para enmarcar el contexto donde se llevaron a cabo las prácticas e implementaciones documentadas en este trabajo. La decisión de realizar este trabajo con este contexto, reside en que la experiencia adquirida con los años, junto con los continuos procesos de mejora, y las líneas de investigación en desarrollo, que reflejan el compromiso de SEDICI para con la institución que lo alberga, así como para con la comunidad científica que lo rodea, siempre en la búsqueda de nuevas instancias de superación.

La Preservación y el Control de calidad

Desde mediados de los 90, en distintas instituciones y empresas del mundo, se ha comenzado a estudiar el problema de la preservación digital [Térmens, 2009]. En este contexto, una gran parte de la comunidad que se encuentra bajo esta rama de investigación, está formada por personal especializado, que pertenece a distintos repositorios institucionales del mundo. Importantes autores como Cristoph Becker, Andreas Rauber, Stephan Strodl, Hannes Kulovitis y Hans Hofman (todos de la Universidad de Viena,Austria), entre otros [Strodl, Becker, Neumayer, & Rauber, 2007], han realizado varios trabajos y participado en proyectos de alcance internacional. Lo que da la pauta de la importancia que ha ido adquiriendo la preservación digital a lo largo de los años, en distintas partes del mundo.

De acuerdo con el rumbo que ha tomado el plano internacional, en el SEDICI, se están llevando a cabo iniciativas en preservación, que han significado grandes avances para el repositorio, con respecto a esta rama de investigación. Incluso se han planteado una serie de aspectos, métricas y métodos a tener en cuenta a la hora de establecer el “estado” de la preservación en un repositorio institucional, utilizando como ejemplo práctico a SEDICI [De Giusti, 2014b]. Para realizar esto, se ha partido por entender a la preservación con una noción, entre otras, similar a la planteada por la UNESCO (United Nations Educational, Scientific and Cultural Organization), que entiende a la preservación digital como:

“El conjunto de prácticas de naturaleza política y estratégica, y las acciones concretas destinadas a asegurar el acceso a los objetos digitales a largo plazo.”

[Biblioteca Nacional de Australia, 2003]

A partir del análisis de esta definición, se puede concluir que la preservación es, en realidad, un conjunto de actividades constantes, que se suelen agrupar bajo una política de preservación seguida por una empresa o institución, que involucra una estrategia para asegurar que los distintos objetos digitales estén disponibles a lo largo del tiempo. Esto es, en el corto, mediano y largo plazo.

Por lo dicho hasta aquí, comienzan a tomar importancia tareas que tienen que ver con diversos aspectos que contribuyen a asegurar la preservación digital, involucrando consideraciones políticas (la formulación de la reglamentación adecuada), económicas (la obtención y disponibilidad de los recursos y el personal especializado necesario) y técnicas (la

formulación de soluciones que ataquen directamente el problema de la preservación). Existe mucha información acerca de las tareas que se involucran en la preservación digital en un repositorio institucional (ver Bibliografía referida al tema: [(Térmens, 2013), (De Giusti, 2014b)]), aunque en este trabajo se hará foco en los aspectos técnicos de la preservación. Por esto, se puede considerar una muy buena reseña de las distintas características a tener en cuenta, a aquellas propuestas por Miguel Térmens[(Fugueras et al., 2011)]:

- Conservar inalterado el bitstream de los ficheros a lo largo del tiempo;
- Mantener la capacidad para renderizar los datos, esto es poseer la capacidad para interpretar el formato en el que están codificados;
- Disponer del software y del hardware necesario para ejecutar los ficheros;
- Mantener el conocimiento para interpretar el contenido de los datos y del software que lo soporta (manuales, órdenes, diccionarios de datos, codificaciones…)

Por lo dicho anteriormente, se comienzan a plantear dos aspectos importantes en el ámbito de la preservación digital (que pueden verse claramente en [De Giusti, 2014b]):

- Las actividades de "*Control de Calidad*" sobre los distintos elementos de un Repositorio Institucional, que buscan asegurar los aspectos antes mencionados, entre otros, y detectar cuando alguno de estos aspectos no se cumple en un elemento.
- Las actividades de "*Curaduría*", que tienen como objetivo reparar los posibles elementos que se han encontrado como inválidos, o que no cumplen con alguno de los aspectos de la preservación antes mencionados.

En este contexto, es en el que se plantean distintas herramientas, tanto para control de calidad, como para actividades de curaduría, que buscan mejorar estos procesos y asegurar que se realicen eficaz y eficientemente. [(Terruzzi et al., 2014), (De Giusti et al., 2013)]

En este trabajo, se plantea una herramienta que se enmarca en la preservación en el repositorio institucional de la UNLP. Primero para realizar controles semiautomáticos de calidad en los distintos elementos del repositorio, después para permitir realizar tareas de curaduría sobre los elementos en los que se localizan problemas (falta de metadatos, bitstreams erróneos, identificadores persistentes inválidos o no asignados, ausencia de "texto completo", etc).

Capítulo 3 | Software DSpace

Introducción

En la actualidad, casi la mitad de los repositorios digitales del mundo están soportados por

el software Dspace [OpenDOAR Chart - Growth of the OpenDOAR Database - Worldwide, s. f.)] y lo mismo sucede en igual o mayor medida con los nuevos

repositorios que se siguen creando día a día. Esto puede atribuirse entre otros motivos a

que es un desarrollo de código abierto, a la funcionalidad que provee, a la gran cantidad de

documentación y experiencias en línea sobre su uso e instalación y en particular a su gran

comunidad de usuarios y desarrolladores que constantemente lo actualiza y expande. [Rauber, Christodoulakis, & Tjoa, 2005]

El software DSpace es un sistema de administración ampliamente utilizado alrededor del mundo para la gestión de repositorios digitales. Presenta muchas funcionalidades que colaboran con la gestión de varios aspectos que deben tenerse en cuenta en los repositorios digitales: la preservación, la gestión de los metadatos, los elementos de visibilidad, la facilidad de búsqueda, la gestión de autores, etc.

Definido por sus propios desarrolladores, DSpace es una plataforma de código abierto que permite que las organizaciones puedan:

- Capturar y Describir material digital a través de un módulo de envíos por flujo de trabajo o una variedad de opciones de importación programáticas
- Distribuir el conjunto de datos digitales en la web, a través de un sistema de buscar y recuperar
- Preservar el material digital, por términos largos de tiempo
- De forma general, puede verse que el sistema es compatible con prácticamente todas las actividades que debería realizar un repositorio digital, un análisis más exhaustivo arrojará que no solo es compatible, sino que es una excelente opción.

Análisis Funcional del Software

La función en la cual hace foco DSpace es brindar soporte para permitir el acceso online a los datos digitales que se gestionan. Para esto, los usuarios pueden acceder a las distintas páginas correspondientes a *Items* o Elementos, que en el sistema representan un conjunto de metadatos de descripción, que se suman al o a los archivos buscados, por lo general, disponibles para ser descargados (si los autores así lo disponen).

Además, DSpace puede procesar los contenidos basados en texto que gestiona, para permitir la búsqueda sobre ”texto completo”. Esto significa que no sólo los metadatos que se proveen para un determinado ítem serán analizados durante una búsqueda, sino que el usuario podrá definir palabras clave, que puedan formar parte del contenido de los archivos del ítem.

Por otro lado, el software permite a los usuarios buscar su propio modo de obtener el contenido apropiado, a partir de varias opciones como: búsquedas personalizadas, navegación a través de APIs, referencias externas utilizando identificadores persistentes como *Handle*. [(Sun, Reilly, Lannom, & Petrone, s. f.)]

También, DSpace ofrece soporte para todo tipo de archivos. Mientras usualmente se utiliza para organizar materiales basados en texto, tales como comunicaciones universitarias, o distintos tipos de tesis electrónicas y disertaciones; existen varios usuarios de la comunidad que lo utilizan para albergar contenido multimedia u objetos de aprendizaje [(DuraSpace, s. f.-f)]. Los archivos que se han subido a DSpace se suelen conocer como “Bitstreams”, esto es porque luego de ser cargados en el software, los archivos son tratados como simples streams de bits en el sistema de archivos anfitrión, sin considerar la extensión o el formato interno de dicho archivo.

Otra funcionalidad a tener en cuenta es su optimización para la indexación con buscadores como Google y Google Schoolar. DSpace añade ciertos metadatos específicos a las páginas que genera para facilitar el trabajo de los buscadores. De hecho, los repositorios DSpace más populares usualmente reciben alrededor del 60% del porcentaje de sus accesos, desde búsquedas en Google. [(DuraSpace, s. f.-l)]

Gestión de Metadatos

De forma general, DSpace considera tres tipos generales de metadatos alrededor de un archivo o elemento en el sistema. Para los **metadatos descriptivos** el software permite varios esquemas de metadatos para describir un item [Duval, Hodgins, Sutton, & Weibel, s. f.]. Por defecto, se provee un conjunto de elementos y calificadores basados en el perfil de aplicación de bibliotecas de Dublin Core calificados [Dublin core, s. f.]. Si bien, se ofrecen varios tipos de elementos y calificadores de metadatos por defecto, el software permite configurar varios esquemas y utilizar los elementos que se consideren necesarios de cada uno.

Para el caso de los **metadatos administrativos,** que incluyen los metadatos de preservación, de procedencia y los datos de la póliza de autorización o la licencia, se brinda soporte a partir de lo albergado en la base de datos. Además, los metadatos de procedencia se copian en los registros Dublin Core (en forma literal) y otros metadatos administrativos (por ejemplo, los tamaños en bytes de los bitstreams o los *MIME Types [Parks, Nelson, & Mitra, s. f.]* de cada uno) también son replicados en registros Dublin Core, para que puedan ser accedidos más fácilmente desde fuera de DSpace.

Los **metadatos estructurales** incluyen información acerca de cómo presentar un elemento, o incluso los bitstreams de un ítem, a los usuarios finales. En DSpace, los metadatos estructurales son bastante básicos; dentro de un ítem los bitstreams pueden ser organizados en distintos bundles, como se describe más adelante. Además, estos bundles pueden tener opcionalmente un bitstream primario, utilizado por el controlador HTML para escoger cuál bitstream enviar primero al navegador. En adición a algún tipo de metadato técnico, un bitstream también posee un ID de secuencia que lo identifica dentro de un ítem, esto se utiliza para producir identificadores persistentes de bitstreams para cada bitstream del sistema. Los metadatos estructurales pueden también persistirse en bitstreams serializados, aunque DSpace no gestiona esta información de forma nativa.

El análisis de los distintos tipos de metadatos y de cómo DSpace los gestiona, los registra y los persiste es mucho más complejo que esta breve exposición y excede los límites de este trabajo. Se puede encontrar más información sobre el análisis de los metadatos, particularmente para tareas de preservación en [(De Giusti, 2014b), (Smith et al., 2003), (Koutsomitropoulos, Solomou, Papatheodorou, & Alexopoulos, 2010), (Kurtz, 2013)]

Versiones e Historia

La primera versión de DSpace fue liberada en noviembre de 2002, a partir de un esfuerzo conjunto por los desarrolladores del MIT y HP Labs (laboratorios de HP) en Cambridge, Massachusetts [(DuraSpace, s. f.-k)]. En marzo De 2004 tuvo lugar el primer DSpace User Group Meeting (DSUG) en Hotel@MIT, y fue ahí donde se produjeron las primeras discusiones concernientes a la comunidad de DSpace y su futura gobernanza.[(Smith et al., 2003)]

La Federación DSpace (DSpace Federation) formó una agrupación flexible de instituciones interesadas, mientras el DSpace Committers Group fue formado poco después, consistiendo en 5 desarrolladores de HP Labs, MIT, OCLC, Universidad de Cambridge, y Universidad de Edimburgo [(DuraSpace, s. f.-g)]. Más tarde se unieron al grupo desarrolladores de la Universidad Nacional Australiana y la Universidad de Texas A&M.

DSpace 1.3 fue lanzado en 2005, y casi al mismo tiempo tuvo lugar el segundo DSpace User Group Meeting en la Universidad de Cambridge. Siguiendo a esto, se celebraron dos mítines menores de grupos de usuarios, el primero en enero/febrero de 2006 en Sydney, y el segundo en abril de 2006 en Bergen, Noruega.

En Julio de 2007 como la comunidad DSpace había crecido mucho, HP y MIT formaron la fundación DSpace una organización sin fines de lucro que provee liderazgo y soporte [Kimpton, s. f.]. De esta forma, en marzo de 2008, la comunidad DSpace liberó DSpace 1.5.

En mayo de 2009, las colaboraciones en proyectos relacionados y la sinergía que comenzaba a volverse importante entre la fundación DSpace y la organización Fedora Commons[Fedora Repository] llevaron a la agrupación de estas dos organizaciones para seguir su objetivo común en una organización sin fines de lucro llamada DuraSpace.

En este contexto, en marzo de 2010, fue liberado DSpace 1.6.

Más tarde, en marzo de 2010 se liberó la versión 1.7, que recibió mejoras hasta julio del 2013, con la versión 1.7.3. No obstante, al mismo tiempo se lanzó la versión 1.8 (noviembre de 2011), cuya última versión, la 1.8.2 data del 24 de febrero de 2012. Ese mismo año, y luego del proyecto que no pudo tener éxito de DSpace 2.0, debido a los costos de su desarrollo, se lanzó DSpace 3.0, que todavía recibe soporte, demostrándose con la última versión hasta la fecha, salida el 24 de febrero de 2015: DSpace 3.4.

En diciembre de 2013, Duraspace propuso una nueva versión del software, adaptada a las tecnologías del momento y que corregía muchos errores encontrados anteriormente: DSpace 4.0. Entre otras cosas, esta versión incorporaba tecnologías como Bootstrap [«Bootstrap », s. f.] y vistas móviles, correcciones en módulos de interoperabilidad como el de Swordv2[«SWORD V2 Specifications », s. f.] y una nueva web service API, compatible con REST [DuraSpace, s. f.-d]. Esta versión de DSpace, es bastante utilizada y continúa recibiendo soporte. La ultima actualizacion hasta el día de la fecha es el 24 de febrero de 2015, la que se considera como la 4.3.

Finalmente, en enero de 2015 se lanzó DSpace 5.0, que ofrece varias herramientas nuevas, entre la que se destaca mucho, la posibilidad de añadir metadatos a todos los objetos del modelo de DSpace (que se explica en la ”*Modelo de DSpace*”). En la actualidad, se encuentra vigente la versión 5.1 del software, que fue lanzada el 25 de febrero de 2015.

Por todo lo dicho en esta sección, en este trabajo se hará referencia a la última versión disponible del Software, la 5.1. Esto es una decisión tomada a partir de la consideración de la posibilidad de utilizar metadatos en todos los objetos del modelo y de la actualidad de la herramienta en los repositorios, especialmente en el repositorio institucional de la UNLP, donde se aplica el desarrollo de este trabajo.

Arquitectura de DSpace

El sistema DSpace está organizado en tres capas, y cada una de ellas consta a su vez de un determinado número de componentes. [DuraSpace, s. f.-b]

La *capa de almacenamiento* es responsable del almacenamiento físico de los metadatos y del contenido. La capa de *lógica de negocios* gestiona las formas de organizar el contenido de los archivos, los usuarios de los archivos (e-personas), la autorización y el flujo de trabajo. La *capa de aplicación* contiene los componentes que comunican con todo lo exterior a la instalación local de DSpace, por ejemplo la interfaz web y servicio de harvesting/recolección de metadatos remotos mediante el protocolo OAIS.[(OAIS), CCSDS Magenta Book. Issue 1.]

Cada capa sólo invoca la capa que tiene debajo de si misma, por ejemplo: la capa de aplicación no debería utilizar la capa de almacenamiento directamente. Cada componente en la capa de almacenamiento y en la de lógica de negocios tiene una API pública definida. La unión de la las APIs de estos componentes suelen ser referenciadas como la API de almacenamiento y la API pública de DSpace respectivamente. Estas APIs son esencialmente clases JAVA, objetos y métodos.

Resulta importante notar que cada capa asume que la otra se está ejecutando en un modo seguro. Aunque la lógica de las acciones de autorización se encuentra en la capa de lógica de negocio, el sistema deriva la autenticación correcta y segura de los usuarios (*personas-e*) a las aplicaciones individuales de la capa de aplicación. De esta forma, si una aplicación insegura pudiese invocar la API pública directamente, podría realizar acciones fácilmente con los permisos de cualquier usuario en el sistema.

La razón para este tipo de diseño reside en que los métodos de autenticación varían ampliamente entre aplicaciones distintas, por lo que tiene sentido dejar la lógica y la responsabilidad de realizar esta operación a tales aplicaciones.

Entender e interpretar la arquitectura de DSpace es importante para realizar desarrollos incrementales y operacionales en el sistema, que puedan ser publicados y reutilizados por la comunidad de usuarios del software. La figura 3.1 muestra la representación de la arquitectura de DSpace, tomada del propio manual del sistema.

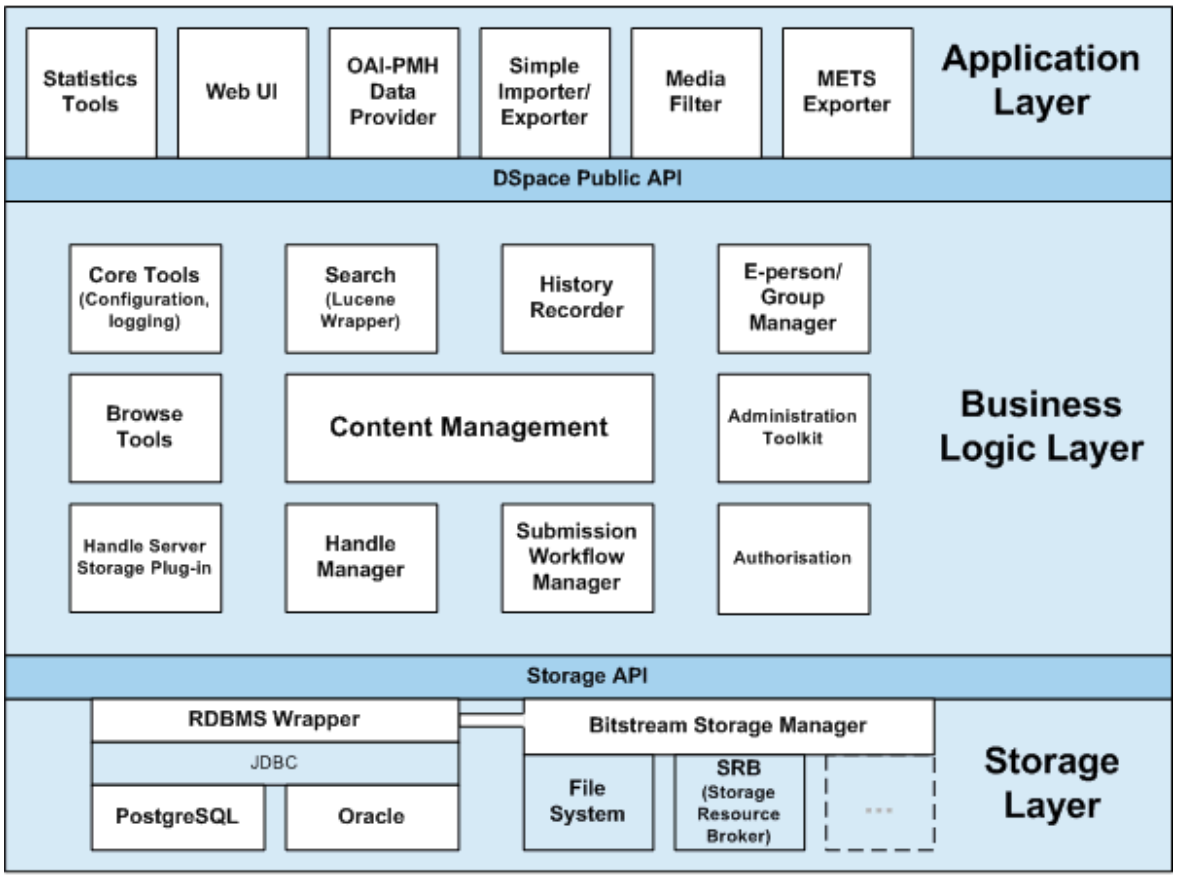

Figura 3.1: organización en capas del software DSpace, tal y como la muestra su documentación.[DuraSpace, s. f.-b]

Módulo *Additions*

El módulo additions es un componente de DSpace añadido a partir de la versión 3.0, como respuesta a una necesidad de la comunidad, de tener un espacio para realizar adaptaciones que involucren cambios a nivel capa de negocios y, en menor medida, capa de almacenamiento en el software, para adaptarlo mejor al dominio de su problema. Esto es una mejora clara con respecto a las versiones anteriores, que sólo ofrecían soporte sobre cambios a nivel de aplicación, por lo que la adaptación del software era costosa y su resguardo resultaba complejo a la hora de realizar actualizaciones.

En este contexto, con DSpace 3.0 se añade el módulo *additions* que (según la documentación del software [DuraSpace, s. f.-a]) debería utilizarse para almacenar los cambios en la dspace-api, los plugins personalizados, los componentes nuevos, etc. Las clases que se ubican en este módulo, alojadas en la ruta [dspace-source]/dspace/modules/additions, sobreescriben las que se encuentran en dspace-api. A su vez, este módulo es referenciado por todas las webapps que se localizan en el directorio [dspace-source]/dspace/modules y por la interfaz de línea de comandos. Por esto, es recomendable localizar todos los cambios a la dspace-api en el módulo additons, para que estén contenidos en un único módulo. Esto facilitaría mucho, la posibilidad de obtener una visión global y un registro de los cambios realizados al software.

Importancia del módulo additions para este trabajo

Se define el módulo additions, porque este trabajo propone una serie de mejoras, adaptaciones y añadidos al software DSpace base, en su versión 5, y los aloja en dicho módulo. De esta forma, no solo se logra añadir funcionalidad al software, sino que se la mantiene encapsulada en un módulo, referenciable desde cualquier punto o componente en la capa de aplicación y de lógica de negocios del sistema.

Modelo de DSpace

La forma en la que los datos son organizados en DSpace intenta reflejar con el software la estructura de una organización [Chapter 2. DSpace System Documentation: Functional Overview», s. f.]. Cada sitio DSpace se divide en *comunidades*, que pueden subdividirse en *sub-comunidades*, permitiendo así trazar un paralelismo con la estructura jerárquica de una institución universitaria típica: universidad, facultades, departamentos, centros de investigación o laboratorios.

La *comunidades* contienen *colecciones*, que son agrupaciones de contenido relacionado. Una colección puede aparecer en más de una comunidad, y está compuesta por *ítems*, los elementos básicos de gestión y que se archivan con el sistema. Cada ítem tiene solo una colección a la que pertenece. No obstante, de forma adicional, un item puede aparecer en distintas colecciones adicionales, aunque su colección de pertenencia es sólo una.

El item es la representación en el modelo de datos, de cada elemento contenido en el repositorio. Se asocian varios datos asociados al ítem como: su última fecha de modificación, la persona que lo subio, la colección en la que se encuentra, etc.

Cada ítem se encuentra asociado a uno o varios *bundles*, mientra que un bundle puede estar asociado sólo a un único ítem.

Los *bundles* son agrupaciones de archivos dentro del item, que separan los diversos tipos de archivos de modo que DSpace pueda tratarlos de forma diferenciada. De hecho, si se nombra un nuevo bundle, DSpace creará el objeto con el archivo correspondiente incluido en dicho bundle. Queda luego la tarea de modificar DSpace para que incorpore tratamientos diferenciados a los archivos del nuevo bundle así creado ya que, claro está, el usuario con permisos normales de DSPACE **solo** podrá ver los archivos contenidos en el bundle ORIGINAL. A su vez, los bundles se relacionan directamente con uno o varios *Bitstreams* y no son más que un conjunto de estos últimos, agrupados bajo cierta lógica. Tal es así, que en la práctica la mayoría de los ítems suelen tener asociados, al menos alguno de estos bundles:

- **Original:** el bundle con el bitstream original depositado (los que se deben seleccionar para evitar listar varios bundle del mismo item)
- **Texto:** texto completo extraído del bitstream original, correspondiente al item
- **Licencia:** contiene la licencia que se subió en conjunto con el item 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* no es más que lo que su nombre indica, una secuencia lógica y ordenada de bits que representan al archivo propiamente dicho. El bitstream es el elemento digital en cuestión. Cada Bitstream se asocia directamente con un formato de Bitstream, esto es así porque los sistemas de preservación implementados (muchas veces como tareas de curación) requieren explícitamente que el usuario que sube cada ítem pueda definir el formato de archivo

con el cual está trabajando para poder conservarlo de mejor forma. Por último, cada formato de Bitstream posee un *MIME type* y un nivel de soporte únicos, estos son los datos reales que suelen utilizar los sistemas de preservación. En la *figura 3.2*, se muestra un esquema del manual de DSpace que representa el modelo de datos utilizado por el sistema.

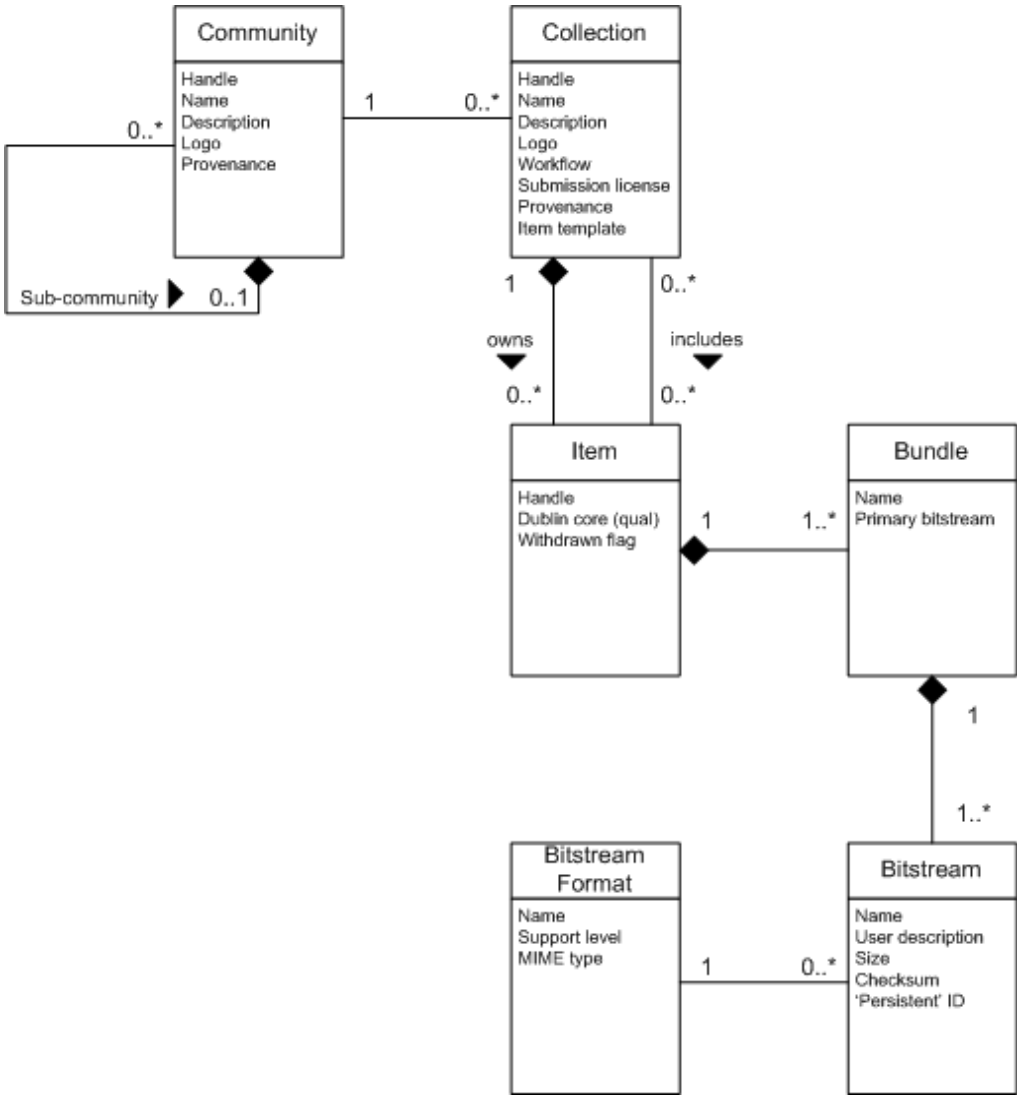

Figura 3.2: modelo de datos de DSpace, tal y como lo muestra su documentación [«Chapter 2. DSpace System Documentation: Functional Overview», s. f.]

Cabe mencionar que en el modelo de la versión de DSpace actual (versión 5) permite que todo objeto DSpace (*DSpaceObject*, los objetos que forman parte del modelo descrito anteriormente) posea metadatos que pueden ser consultados [(DuraSpace, s. f.-i)]. Esta idea está descrita en la nueva documentación del software y quedan plasmadas en las distintas interfaces públicas de clases como la clase DSpaceObject.

En conjunto con este modelo de objetos se deben considerar ciertas clases que son parte del núcleo del sistema, en la capa de lógica de negocio y que son utilizados en varios lugares del sistema. Algunas de estas clases son: el Contexto, el Sitio, las E-Persons y los Grupos Especiales.

Contexto

La clase Contexto es central en la operación de DSpace. Cualquier código que necesite utilizar alguna API en la capa de lógica de negocio, debería crear primero un objeto Contexto.

Los objetos contexto están involucrados en la mayoría de las llamadas a métodos y en los constructores de los objeto. De esta forma, el método u el objeto instanciado tiene acceso a la información acerca de la operación actual. Esto es porque, en el momento en donde el objeto Contexto es construído, se inicializa automáticamente la siguiente información: una conexión a la base de datos y una cache de la API de objetos de gestión de contenidos. A su vez, la siguiente información también se mantiene en el objeto context, aunque es responsabilidad de la aplicación que lo crea completarla correctamente durante la instanciación: el usuario autentificado actual, de existir; los grupos especiales a los que pertenece el usuario; cualquier información extra de la capa de aplicación que sea considerada importante para la instanciación; etc.

E-persons y Grupos

DSpace mantiene un registro de los usuarios con la clase *org.dspace.eperson.Eperson*. La clase posee métodos para crear y manipular una EPerson tales como los métodos get y set para nombres y apellidos, correos, passwords, etc. Existen métodos *find* para encontrar a una persona por su correo (se asume que es único), o para encontrar todas las Personas-E (EPeople) en el sistema.

Los *Grupos* son simplemente listas de objetos EPerson. Además de los miembros, los objetos de la clase Grupo tienen sólo un atributo: un nombre. Los nombres de grupo deben ser únicos y se pueden añadir y eliminar objetos persona a través de los métodos *addMember()* y *removeMember().* El objeto contexto de sesión posee una lista de los IDs de grupo a los que el usuario de la sesión pertenece. (Ver mas en: (DuraSpace, s. f.-c))

Grupos especiales

Se asume que todos los usuarios son parte del grupo público (ID=0). Los *DSpace admins* (ID=1) son parte de todos los grupos automáticamente. El objeto Contexto también mantiene una lista de los grupos especiales, de los que se debe comprobar en primer lugar si el usuario actual pertenece o no.

Site

Representa la raíz del sistema de archivos de DSpace. Por defecto, el sufijo *Handle* “0” identifica al sitio. Es parte del modelo de datos de dspace al ser una subclase de DSpaceObject. Mediante los distintos métodos en la clase *org.dspace.content.Site*, se puede acceder a datos que el sistema resguarda correspondientes a todo el sitio, independientemente de la colección, comunidad o item que se este navegando. Son ejemplos de esto la URL, el ID o el prefijo Handle del sitio.

Utilización del modelo de DSpace en este trabajo

Como ya se dijo, entender el funcionamiento del sistema y de su modelo, es importante para realizar cualquier aporte de cualquier ámbito a la comunidad: pruebas, documentación, desarrollos, plugins, adicionales, etc.

En este trabajo, se plantea una herramienta que utiliza el contexto DSpace como propio, debido a que se añade como un componente más del sistema. Durante el desarrollo de este trabajo, se realizan referencias a distintos objetos y clases del modelo aquí descrito, debido a que el mismo representa el conocimiento en el dominio que se ha requerido para entender las

necesidades y proponer las soluciones requeridas, que se tratan en en este trabajo. Por esto, se verán sucesivas referencias a esta sección, cuando se consideren necesarias.

Curation/ Tareas de Curación

Desde la versión 1.7, DSpace soporta la ejecución de tareas de curación. Por defecto el software incluye varias tareas útiles, pero el sistema fue inicialmente diseñado para permitir que se añadan nuevas tareas a lo largo de las distintas versiones, tanto las de propósito general que la comunidad de usuarios puede definir, como las escritas de forma local, por los desarrolladores.

Definición de Tareas de Curación

Puede definirse a una tarea de curación como una forma sencilla y extensible de gestionar ciertas operaciones sobre los contenidos de un repositorio [Bossons, Rodgers, & Shepherd, 2011]. Estas operaciones se conocen como “tareas”, y pueden operar en cualquier DSpaceObject (definido en la sección anterior), lo que significa en el sitio entero, las comunidades, las colecciones, los ítems,etc. Las tareas suelen trabajar sobre un único tipo de DSpaceObject (típicamente un item), y en ese caso simplemente ignorarán los otros tipos (probablemente utilizando la sentencia *task.skip* que brinda el entorno). Las “tareas” pueden ejecutarse de manera interactiva o no interactiva, y se activan desde interfaces distintas (Por ejemplo: la interfaz web del adminsitrador, XMLUI, y la consola del sistema).[(DuraSpace, s. f.-d)]

Por todo esto, este marco de trabajo permite, por ejemplo, que los sitios DSpace tengan una herramienta más que brinda la posibilidad de personalizar el comportamiento de su repositorio, sin tener que alterar el código fuente de DSpace.

Modelo de Tareas de curación

Una tarea de curación es una clase java que puede contener cualquier tipo de código correspondiente a una operación, pero debe cumplir con dos propiedades:

Primero, debe proveer un constructor sin argumentos, para que pueda ser cargado con el *PluginManager* de Dspace [(DuraSpace, s. f.-h)]. Esto es porque las tareas, dentro del contexto de DSpace, son vistas como plugins con nombre, donde el nombre de la tarea se corresponde con el del plugin.

En segundo lugar, las tareas de curación deben implementar la interface “*org.dspace.curate.CurationTask*”, que propone Dspace [(«Curation Task Cookbook - DSpace - DuraSpace Wiki», s. f.)]. Esta interface es, en su mayor parte, una interfaz de etiquetado, por lo que sólo requiere que unos pocos métodos de alto nivel sean implementados. De esta forma, una tarea de curación personalizada debe ser una clase java que implemente los siguientes métodos:

- *Init(Curator curator, String taskId)*: se inicializa la tarea, se cargan sus parámetros (en caso de tenerlos) y se informa a la tarea del *curator* que la ha invocado (el *curator* es a las tareas de curación, lo que el pluginManager es a cualquier plugin).
- *perform(DSpaceObject dso)*: Es el método más importante a implementar, y es donde la tarea ejecuta una acción determinada sobre el DSpaceObject con el cual se la invocó. Como resultado se debe devolver un valor entero, un código que describe una de cuatro posibles condiciones: 0 (Éxito, la tarea se completó exitosamente), 1 (Fallo, la tarea falló, queda en la tarea definir qué se cuenta como un fallo), 2 (Omitido, la tarea

no pudo ejecutarse en el objeto, quizás por no ser de la clase adecuada), -1 (Error, la tarea no se pudo completar debido a un error).

- *perform(Context ctx, String id):* Este método brinda la posibilidad de ejecutar la tarea de curación sobre un objeto a partir de su ID. En general, es recomendable implementar el *perform()* que recibe un dso y si se quiere re-implementar este, obtener el DSpaceObject por su id e invocar el otro *perform().*

Cabe destacar, que las nuevas tareas pueden extender la clase abstracta AbstractCurationTask (org.dpsace.curate.AbstractCuration), que define los métodos *init()* y *perform(Context ctx, String id)*, y asegura que una tarea de curación se construya correctamente, requiriendo que sólo implemente el método *perform(DSpaceObject dso)*. [(DuraSpace, s. f.-d)]

Trabajos sobre curación

La idea de implementar tareas de curación que colaboren de alguna forma con alguna funcionalidad del repositorio, principalmente para controles de calidad y tareas de preservación, ha surgido en varios repositorios del mundo y actualmente es una línea de trabajo usualmente escogida por los usuarios del dominio de repositorios digitales.

En el contexto internacional, se han realizado diversos avances en esta línea de trabajo.

Desde comienzos del 2007, se empezó a trabajar sobre la importancia de las tareas de curación en los repositorios digitales. En la Universidad de Cornell se planteó un método original para desarrollar tareas conjuntas de curación en ciertos metadatos que se gestionaban en el repositorio. [(Steinhart & Lowe, 2007)]

Más tarde, en el 2008, se estudiaron las tareas de curación de datos, desde un punto de vista amplio y generalizado, en un caso de estudio planteado por la Universidad Johns Hopkins. [(Choudhury, 2008)]

Luego, en el 2010, diversos estudios permitieron la realización de un análisis de la documentación existente sobre los métodos de curación, que pueden verse en varios artículos publicados en la época, como por ejemplo “Selected Internet Resources on Digital Research Data Curation“ [(Westra, Ramirez, Parham, & Scaramozzino, 2010)] o en el libro “More Technology for the Rest of Us: A Second Primer on Computing for the Non-IT Librarian” [(Nancy, 2010)].

Después, en el año 2011, la Universidad de Porto llevó a cabo un experimento con tareas de curación, comenzando a implementarlas de manera manual, y entendiendo la necesidad de plantear una posible automatización de las mismas como trabajo futuro [(Silva, Ribeiro, & Lopes, 2011)]. Ese mismo año, Durante la Sexta Conferencia Internacional de Repositorios Abiertos, se definieron los lineamientos a seguir para implementar tareas de curación en DSpace, que permiten automatizar ciertos aspectos. (Bossons et al., 2011)

Estos lineamientos continúan reflejados en trabajos de la actualidad, como por ejemplo: los desafíos de la preservación digital propuestos en la Universidad Técnica de Texas. [(Perrin, Winkler, & Yang, 2015)]

Con el contexto internacional definido en los párrafos anteriores, en el Repositorio Institucional de la UNLP se llevaron a cabo experimentos y se propusieron soluciones orientadas a tareas de curación. En el 2013 se propusieron tareas de curación orientadas a distintas funcionalidades y mejoras posibles a la gestión de curación planteada por DSpace. Más tarde, durante el 2014 se realizó la implementación de un sistema de evaluación

automática de metadatos de preservación, utilizando el mecanismo de curación de DSpace. [(Terruzzi et al., 2014)]

Importancia de las tareas de Curación para este trabajo

Como se pudo apreciar en la sección anterior, en el ámbito de los repositorios digitales, la utilización de tareas de curación para resolver problemas de control de calidad, tiene varios precedentes. Es por esto que para solucionar varias de las necesidades que plantea este trabajo (ver Motivación y Objetivos, capítulo 1), se optó por una solución que se implemente mediante tareas de curación para DSpace, versión 5. Los modelos planteados y las soluciones propuestas surgirán a partir del conocimiento sobre tareas de curación aquí mencionado y por esto, se hará referencia a esta sección cuando se crea necesario.

Capítulo 4 | Enfoque de la Herramienta de Validación

Análisis del problema

Para la lista de objetivos específicos planteada (*Capítulo 1*), se comenzaron a examinar distintas alternativas de solución, a partir del estudio del dominio. Esto se basó en la idea de buscar una herramienta configurable y ortogonal, además de lo suficiente expresiva como para abordar los conceptos del ámbito de los repositorios.

Debido a la experiencia del equipo del repositorio institucional de la UNLP (ver capítulo 2), se utilizó un método de análisis de dominio planteado en “Organization Domain Modeling guidebook”[(Simos, 1995)], donde se proponen tres pasos para analizar los dominios: planificar el dominio, modelar el dominio y, opcionalmente, construir la solución a partir de los sistemas que ya plantean posibles soluciones para problemas similares. Entre los resultados de la recopilación de soluciones similares se pueden listar:

- En el contexto de un repositorio que se dedica a albergar documentos que sólo pertenecen a un dominio específico: medicina, para resolver el problema de realizar consultas específicas por usuarios expertos, se implementó un lenguaje específico de dominio para realizar consultas con fases múltiples.[(Madaan & Bhalla, 2013)]
- El desarrollo de *lava*, un lenguaje específico para producir gramáticas. Donde la experiencia en el uso y la implementación de este lenguaje, ha demostrado que los lenguajes de propósito especial para la producción de gramáticas pueden traer varios beneficios como mejorar la cobertura, la simplicidad, la manejabilidad, las estructuras de testeo, etc. [(Sirer & Bershad, 2000)]
- Para el problema propiamente dicho de la definición de un dominio se creó una solución con un lenguaje específico de dominio: FDL. Este surgió para implementar la descripción de features, necesarias para el diseño de lenguajes.[(van Deursen & Klint, 2002)]
- Para el dominio de la manipulación de imágenes se diseñó y estructuró Envision, como una extensión de Scheme, un lenguaje que daba soporte para digitalización. [(Stevenson & Fleck, s. f.)]

Los trabajos estudiados en los párrafos anteriores, plantean la aplicabilidad de soluciones en el contexto del desarrollo de lenguajes. Además, estas soluciones no solo fueron construidas con lenguajes de uso general, sino con herramientas más acotadas como los lenguajes específicos de dominio. En este capítulo se definen una serie de conceptos teóricos correspondientes al ámbito de los lenguajes específicos de dominio, luego se estudia la aplicabilidad al problema planteado,de distintos patrones de decisión que brindan indicios de la orientación de una posible solución. Después, se definen distintas herramientas del ámbito de los DSLs, que se utilizan en el capítulo siguiente, para formular una solución al problema.

Concepto de Lenguajes Específicos de Dominio

Los Lenguajes Específicos de Dominio o DSL son lenguajes que han sido optimizados para una clase determinada de problemas, denominada dominio, y que se basan en abstracciones que están fuertemente relacionadas con el dominio para el que fueron construidos [(Voelter, 2013). Formalmente, se define a los DSL como : “*un lenguaje de programación o una especificación de lenguaje ejecutable que ofrece a través de anotaciones y*

abstracciones apropiadas, un poder de expresividad centrado en, y usualmente restringido a, un dominio particular de un problema."[(Deursen, Klint, & Visser, 2000)]

Los DSL son usualmente *declarativos*, es decir que pueden ser vistos como lenguajes de especificación y como lenguajes de programación. Un lenguaje declarativo se contrapone con uno imperativo [(Tucker, 2003)] a partir de que esta basado en el desarrollo de programas especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la computadora qué es lo que se desea obtener o qué es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con *transparencia referencial [(Akhmechet, 2010)]*. Por esto es que, en comparación a sus contrapartes imperativas, los lenguajes declarativos proveen un nivel de programación más alto y abstracto que lleva a programas más confiables y mantenibles [(Hanus, 2007)]. Incluso, por ser lenguajes declarativos, muchos DSL son soportados por compiladores que generan aplicaciones a partir de programas DSL. En este caso, los compiladores pueden verse como generadores de aplicaciones y los DSL como lenguajes especificadores de aplicaciones. [(Bentley, 1986), (Cleaveland, 1988)]

Tipos de DSL

Los DSL suelen clasificarse de varias formas distintas debido a la gran diversidad que existe entre ellos. Según su forma de implementación se pueden clasificar en: [(Deursen et al., 2000)]

- **DSL Internos:** los DSL internos son formas particulares de utilizar un lenguaje de programación anfitrión, para darle la forma de un lenguaje específico. En este enfoque se pueden encontrar DSL realizados en Ruby, Lisp, Java y C#, entre otros. Este tipo de DSL son referenciados a menudo como DSLs embebidos o interfaces “Fluent”, debido a los patrones de implementación que se suelen utilizar en ellos.
- **DSL Externos:** los DSL externos tienen su propia sintaxis personalizada y se debe escribir un parser completo para procesarlos. Muchas configuraciones XML han terminado siendo DSL externos, aunque la sintaxis XML no suele ser adecuada para este propósito.

Patrones de decisión para plantear la herramienta como lenguaje

Existen una serie de patrones que caracterizan a un problema que puede requerir el diseño de este tipo de soluciones. Para corroborar que el enfoque estudiado seguía los lineamientos de los trabajos realizados en el tema, se describe el grado de aplicabilidad de estos patrones definidos como *patrones de decisión.* [(Mernik, Heering, & Sloane, 2005)]

Patrones de decisión

Patrón de notación

La disponibilidad de notaciones específicas del dominio apropiadas (nuevas o existentes) es un factor decisivo en la aplicabilidad de una solución con un DSL. Este tipo de patrón de notación suele dividirse en dos subpatrones:

- Transformación de la notación visual en notación textual: existen beneficios en transformar una notación visual disponible en su formato de texto correspondiente, esto

se puede lograr a partir de una composición sencilla de programas o especificaciones extensas, permitiendo que después se aplique el patrón AVOPT mencionado más adelante. Para el caso específico de los repositorios digitales, en el capítulo siguiente se analiza el dominio y, por consiguiente, se establecen varios modelos que buscan resumir los conceptos específicos y delimitar las características de la solución.

- Añadir una notación amigable para el usuario para una API que ya haya sido desarrollada o transformar esta API en un DSL.

Este patrón de decisión es muy aplicable a el problema tratado debido a que ya se cuenta con ciertas herramientas de validación de la preservación [(Terruzzi et al., 2014)] y un objetivo específico de este trabajo es transformar estos mecanismos en amigables con un usuario que no posea conocimientos técnicos en preservación.

Patrón AVOPT

AVOPT es el acrónimo para análisis específico del dominio, verificación, optimización, paralelización y transformación. En ciertos casos, los programas escritos con un lenguaje de uso general (GPL) no son fáciles de exponer a este patrón, debido a que presentan códigos demasiado complicados o de estructura indefinida. El uso de un DSL apropiado puede hacer aplicable estas operaciones.

Por esto, un patrón de decisión claro a la hora de evaluar una solución, es la necesidad de realizar las operaciones AVOPT en un dominio determinado.

En el caso puntual del problema tratado en este trabajo, se podría ver como una aplicación del patrón AVOPT, la necesidad de mejoras planteadas a los mecanismos de control actuales que el equipo del repositorio ha notado a través del tiempo y que han ido quedando plasmadas en algunos trabajos. [(Terruzzi et al., 2014)]

Patrón de Automatización de Tareas

El hecho de que se piense en automatizar tareas que existen en el dominio y que esto sea complejo y repetitivo en ciertos lenguajes de programación es un patrón decisión que se ha visto en una gran cantidad de soluciones orientadas a lenguajes específicos de dominio. La necesidad de automatizar tareas es, en muchos casos, el objetivo principal de aquellos que buscan una solución con DSL. Esto se da debido a que una de las principales ventajas que poseen estas resoluciones, es la capacidad de automatizar ciertos aspectos del dominio, con los generadores de aplicación o ”*compiladores de DSL*”. (El módulo de expresiones planteado en el capítulo 7 es un ejemplo de esto).

Patrón de línea de producción

Este patrón tiene cierta superposición con el ya explicado patrón de automatización de tareas y con el patrón de front-end del sistema que se explicará más adelante. Cuando se posee una línea de producción de software, cuyos productos comparten una arquitectura similar y son construidos con el mismo conjunto de elementos básicos. El uso de un DSL puede resultar de mucha ayuda ya que posiblemente facilite la especificación de estos productos.

Subjetivamente, se puede tomar a las tareas de control de calidad sobre repositorios digitales, como productos que siguen cierta línea de producción; esta relación podría llegar a ser la aplicabilidad de este patrón de decisión.

Patrón de representación y recorrido de estructuras de datos

Cuando se quieren crear nuevos programas que utilizan estructuras de datos complejas, propias del dominio con el que se esta trabajando, suele ser dificultoso realizar escribir y mantener estos programas. La necesidad de tener que expresar estas estructuras de una manera más amigable, es un patrón de decisión recurrente en una solución con lenguajes específicos de dominio.

En el caso de los repositorios digitales, expresar los modelos de los repositorios y las restricciones de dominio que estos imponen, suele ser una tarea dificultosa (se verá más adelante). Realizar programas que contemplen estas estructuras, por ejemplo, para tareas de control de calidad, involucra una gran cantidad de tiempo y conocimiento en el dominio, por lo que este patrón también se hace presente. [(De Giusti et al., 2013)]

Patrón de front-end del sistema

La necesidad de poseer o diseñar un front-end en un sistema para manejar su configuración y adaptación, es un patrón de decisión que suele mostrarse. Cuando se requiere una extensa configuración de un sistema para brindar una solución, se suele establecer un DSL que facilite esta configuración y actúe como front-end del sistema.

En el caso específico de los repositorios digitales y de las tareas de control de calidad, la necesidad de un mecanismo que facilite la configuración de algunos mecanismos de validación ya se había planteado con anterioridad.[(Terruzzi et al., 2014)]

Patrón de interacción

Este patrón suele darse en conjunto al mencionado en el párrafo anterior (patrón de front-end del sistema). Cuando se tiene una interacción frecuente con una aplicación y dicha interacción es textual o basada en algún sistema de opciones, suele suceder que los datos ingresados de esta forma son complejos y repetitivos, tener un DSL que permita describir estos datos es una buena posibilidad.

En este contexto, la necesidad de que la herramienta de validación sea configurable y manipulable es un objetivo específico de este trabajo y es algo que está justificado en los trabajos anteriores del equipo del repositorio. [(Terruzzi et al., 2014), (De Giusti et al., 2013)]

Planteo de la Herramienta como Lenguaje

Como se puede ver en la sección anterior, existe cierta aplicabilidad de los patrones planteados al problema de control de calidad, actividades de curaduría y preservación en repositorios digitales. Además, ya se han brindado una serie de soluciones a problemas similares utilizando DSLs (mencionadas en la sección anterior), y en los mismos también son aplicables estos patrones de decisión. Por esto, se propuso la orientación de la herramienta de validación en la línea de estudio de la construcción de lenguajes específicos de dominio.

A partir de aquí aparecen en el planteo de una solución, varios mecanismos que tienen que ver con los lenguajes específicos de dominio y su línea de investigación. A continuación se definen los mecanismos que son utilizados luego *(capítulo 5)*, en la descripción y especificación de una solución.

Mecanismos de desarrollo de lenguajes específicos de dominio

Sintaxis abstracta

La sintaxis abstracta de un lenguaje es una estructura de datos que contiene la información principal en un programa, pero sin ninguno de los detalles de anotación contenidos

en la sintaxis concreta: palabras claves y símbolos, la disposición (por ejemplo, los espacios en blanco) y los comentarios no suelen ser incluidos en la sintaxis abstracta.

La sintaxis abstracta suele definir reglas que determinan si un modelo escrito en ese lenguaje es válido para lo que él mismo define. Se suele recomendar que se defina primero la sintaxis abstracta de un lenguaje porque es la base sobre la cual los otros artefactos del lenguaje se van a definir.[(Mernik et al., 2005),(Fowler, 2005), (Louden, 2004)]

La sintaxis abstracta se enfoca en las relaciones estructurales existentes entre los conceptos del lenguaje y no en describir su semántica. Por esto, el modelo de esta sintaxis debe describir las reglas para determinar si un modelo esta bien formado: es sintácticamente válido. Esto proporciona una descripción más detallada de las reglas sintácticas del lenguaje (en vez de describir conceptos y relaciones).

Existen varios enfoques destinados a modelar la sintaxis abstracta de un lenguaje; los más comunes suelen ser la definición y construcción de árboles de sintaxis abstracta y la implementación de un metamodelo. En general, para definir un DSL se utilizan ambos enfoques: se hace que los parsers construyan un modelo semántico a partir del árbol de sintaxis abstracta, resultado del análisis léxico de un programa en ese DSL. Este proceso puede variar de acuerdo al tipo de DSL con el que se esté trabajando. Para comprender mejor este proceso deben definirse primero los dos enfoques.(Fowler, 2010)

Árboles de sintaxis abstracta

Un árbol de sintaxis abstracta (AST), o simplemente un árbol de sintaxis, es una representación de árbol de la estructura sintáctica abstracta (simplificada) del código fuente escrito en cierto lenguaje de programación. Cada nodo del árbol denota una construcción que ocurre en el código fuente.

Un árbol de sintaxis abstracta captura la estructura esencial de una entrada en forma de cadena y le da forma de árbol, mientras omite detalles sintácticos innecesarios. Los ASTs pueden ser distinguidos de los árboles de sintaxis concreta debido a que omiten los nodos del árbol que representan las marcas de puntuación tales como las comillas para terminar las declaraciones o las comas que separan los argumentos de una función. Los ASTs también omiten los nodos del árbol que representan producciones unarias. Esta información es directamente representada en los ASTs por la estructura del árbol.

Los ASTs pueden ser creados con parsers hechos a mano o mediante código producido por generadores de parsers y son en general construidos mediante razonamiento inductivo (bottom-up).

Cuando se diseñan los nodos de un árbol, una decisión común de diseño es determinar la granularidad de la representación de un AST. Esto significa decidir si todas las construcciones del lenguaje fuente van a ser representadas como tipos diferentes de nodos en el AST, o si algunas van a representarse con un tipo de nodo común y se van diferenciar usando un valor. Un ejemplo de la elección de la granularidad de la representación es determinar cómo representar las operaciones aritméticas binarias: una opción es tener un nodo del árbol que represente una operación aritmética binaria, el cual posee en uno de sus atributos la operación adecuada (ej: “*”). La alternativa sería tener un nodo del árbol por cada una de las operaciones binarias existentes. En lenguajes orientados objetos, esto resultaría en tres clases tales como: SumaBinaria, RestaBinaria, MultiplicacionBinaria, etc. con una superclase abstracta Binaria. Esta segunda opción suele preferirse cuando existe algún comportamiento asociado a los nodos del árbol. (Ver más en: [(Jones, 2003)])

Modelo semántico

En el contexto de DSL un modelo semántico es una representación de todos los conceptos que el lenguaje puede describir. De esta forma un programa en ese lenguaje, definirá valores particulares del esquema definido por el modelo semántico. Por esto se puede ver al modelo semántico como la librería o el framework que es utilizado por los programas escritos en el DSL.

De acuerdo con el tipo de DSL (externo o interno) que se esté diseñando existen varias formas de representar el modelo semántico: objetos en memoria, estructuras de datos con funciones que las modifiquen o incluso un modelo guardado en una base de datos relacional.

El modelo semántico debería ser diseñado teniendo en cuenta el propósito del DSL. Más aún, el modelo semántico debería poder ser utilizado sin la necesidad de la existencia de un DSL, se podría implementar un modelo que responda al mismo, por ejemplo desde una interfaz de consulta mediante comandos. Esto aseguraría que el modelo capture toda la semántica del dominio del problema y permitiría que se pueda testear sin necesidad de implementar un parser.

Un modelo semántico es usualmente diferente a un árbol de sintaxis debido a que tienen propósitos distintos. Un árbol de sintaxis se debe corresponder con la estructura de los programas construidos con el DSL. A pesar de que un AST puede simplificar y reorganizar los datos de entrada, sigue manteniendo la misma forma fundamental.

El modelo semántico, por otro lado, se basa en lo que debe realizarse con la información de un programa DSL. Este será a menudo una estructura diferente, incluso ni siquiera tendrá una estructura de árbol. Existen ocasiones donde un AST es un modelo semántico efectivo para un DSL, pero suelen ser excepcionales.

La teoría de lenguajes tradicional ([(Louden, 2004)]) no suele utilizar un modelo semántico y esto suele citarse como parte de las diferencias entre trabajar con DSL y con lenguajes de propósito general([(Fowler, 2010; Mernik et al., 2005)]). Un árbol de sintaxis usualmente es una estructura muy útil para la generación de código base en un lenguaje de propósito general, por lo que no suele optarse a menudo por poseer un modelo semántico distinto, que se derive del AST. No obstante, muchos modelos son utilizados como representaciones intermedias (pasos intermedios en la generación de código).

El modelo semántico a menudo puede preceder al DSL. Esto suele pasar cuando se cae en la cuenta de que alguna porción del dominio del problema se puede definir mejor al construir el DSL que desde una interfaz orientada a comandos regulares. Alternativamente, se puede construir un DSL y un modelo semántico al mismo tiempo, iterando sobre la especificación del dominio y refinando tanto las expresiones del lenguaje, como la estructura del modelo semántico.

Para resumir el concepto de un modelo semántico, puede decirse que este modelo debe mapear los conceptos del dominio a los elementos del lenguaje, por lo que la definición del modelo semántico suele ser el mejor momento para validar la solución del problema, debido a que se cuenta con toda la información y las estructuras necesarias para expresar y ejecutar las validaciones. En particular, es útil para validar que los modelos formados cumplan con las reglas de negocio y la delimitación del dominio antes de ejecutar un intérprete o de generar el código correspondiente.

Generación de un modelo semántico: el trabajo de parseo

El proceso de generar un modelo semántico suele ocurrir en el paso de parseo. En este momento aparecen las diferencias entre los DSLs externos e internos, tanto en aquello que se

parsea como en la forma en la que el parseo se realiza. Los dos estilos de DSL producirán el mismo tipo de modelo semántico, por lo que no hay razón para no tener un único modelo semántico que pueda construirse de ambos modos [(Fowler, 2010; van Deursen, Klint, & Visser, 2000)].

Con un DSL externo, existe una clara separación entre los programas construidos, el parser y el modelo semántico. Los programas son escritos en un lenguaje separado, el parser lee estos programas como si fuesen una secuencia de cadenas y construye el modelo semántico (directamente o armando los árboles de análisis sintáctico antes).

Con un DSL interno es más probable que las capas se mezclen. Suele utilizarse una capa explícita de objetos (*Expression Builders*) [(Fowler, 2010)] con el objetivo de proveer las interfaces fluidas (fluent) necesarias para actuar como el lenguaje. Los programas DSL se ejecutarán entonces mediante la invocación de métodos del Expression Builder que generará el modelo semántico. Por esto, en un DSL interno, el parseo de los programas DSL se realiza mediante una combinación del parser del lenguaje anfitrión y los *Expression Builders*. Esto lleva a un aspecto importante en la construcción del modelo semántico: el concepto de “parsing” puede resultar algo difícil de utilizar en el contexto de un DSL interno. No obstante esta forma de pensamiento suele ser útil para explicar los paralelismos entre los procesos de los DSL internos y externos.

Con el parseo tradicional, se toma una cadena de texto, se ordena el texto en un árbol de parseo (AST) y se procesa dicho árbol para producir una salida útil. Cuando se parsea un DSL interno, la entrada es una serie de llamados a funciones. Esta entrada se ordena de la misma forma en una jerarquía (usualmente implícita en la pila de ejecución) con el objetivo de producir una salida útil.

De lo dicho en el párrafo anterior se puede notar que en varios casos el parseo en DSL no requiere manejar el texto de los programas directamente. En un DSL interno, el parser del lenguaje anfitrión es quien maneja el texto y el procesador del DSL es quien maneja las construcciones del lenguaje.

La aplicación de los mecanismos anteriores, permite definir una solución orientada a lenguajes específicos de dominio. No obstante, también existen herramientas para asistir en el desarrollo de una solución, a continuación se definen algunos patrones que luego serán utilizados y referenciados durante la implementación de la solución. (Capítulo 6).

Patrones y Herramientas de implementación en DSL

Con la descripción ya formulada, se comenzaron a estudiar distintas alternativas de implementación. Existen distintos patrones de implementación en el ámbito de los lenguajes específicos de dominio, que sirvieron de guía para el estudio de varias herramientas o alternativas a considerar durante la fase de implementación. En base a esto, se pensó en una herramienta que pudiera resolver los casos de uso antes planteados y cuyo tiempo de aprendizaje, utilización e implementación, fuera adaptable al alcance de esta tesina. A continuación, se listan los diferentes patrones y las herramientas que los aplican, seguido de las ventajas y desventajas que conllevan. [(Mernik et al., 2005)]

Intérprete

Cuando se utiliza un *intérprete*, las construcciones del DSL son reconocidas e interpretadas utilizando un ciclo estándar *de buscar-decodificar-ejecuta*r. Este enfoque es apropiado para aquellos lenguajes que poseen un carácter dinámico o en donde la velocidad

de ejecución no es un problema. Las ventajas del desarrollo de un intérprete sobre otros enfoques como un compilador son su mayor simplicidad, mejor control sobre el ambiente de ejecución y mayor facilidad de extensión. Ejemplos de herramientas que se desarrollaron de esta forma son: ASTLOG [(Crew, 1997)], Service Combinators [(Cardelli & Davies, 1999)].

Generador de aplicación/ Compilador

En este patrón los términos del DSL son traducidos en términos del lenguaje base y en llamados a librerías. Esto permite la realización de un análisis estático completo en el script o programa que utiliza el DSL, sobre la sintaxis del mismo. Por esto, los compiladores de DSL son a menudo llamados generadores de aplicación. Como ventajas lógicas de este patrón se pueden apreciar sus mecanismos de control de errores, la velocidad de ejecución que pueden ofrecer a los programas generados y que suelen no ser intrusivos con respecto a la gramática, por lo que se pueden procesar todos los términos dependendientes del dominio sin problemas. Las desventajas suelen residir en las dificultades para desarrollarlos, expandirlos y aprenderlos. Ejemplos de DSL que utilicen este patrón son: ATMOL ([(van Engelen, 2001)]); lava[(Sirer & Bershad, 2000)], etc.

Preprocesador

Cuando se aplica un preprocesador los términos del DSL se traducen directamente en términos de un lenguaje existente (que se denomina como lenguaje anfitrión o base). Como resultado de esto, el análisis estático se limita al que se realiza con el procesador del lenguaje base. Al aplicar un preprocesador existen sub-patrones que pueden tenerse en cuenta, tales como:

- Macro-procesamiento: Expansión de las definiciones de las Macros. Ejemplo: S-XML [(Kiselyov & Lisovsky, 2002)].
- Transformación código-a-código: El código fuente del DSL se transforma (traduce) en código fuente del lenguaje base. Ejemplo: ADSL,AUI, TVL.[(Gray & Karsai, 2003)]
- Procesamiento en línea (Pipeline): Se tienen procesadores que van manejando sucesivos sub-lenguajes de un DSL y traduciendolo poco a poco, hasta llegar al código generado en el lenguaje base. Ejemplo: CHEM.[(Bentley, 1986)]
- Procesamiento léxico: con el uso de preprocesadores, solo es necesario un análisis léxico sencillo, ya que lo demás se generará a partir de las construcciones del lenguaje base. Ejemplo: SSC. [(Buffenberger & Gruell, s. f.)]

Lenguaje Embebido

En el enfoque de lenguaje embebido, un DSL se implementa mediante la extensión de un lenguaje de uso general existente (el lenguaje anfitrión), a partir de la definición de tipos abstractos de datos específicos y operadores. Esto se diferencia del patrón anterior porque no se implementa un procesador, sino que se utilizan los términos del lenguaje anfitrión para construir los términos del dominio. Por esto, el nuevo lenguaje tendrá toda la potencia del lenguaje anfitrión, pero puede verse a simple vista que la curva de aprendizaje del mismo será compleja, debido a que para utilizar el DSL, se requerirán conocimientos en el lenguaje base. Son ejemplos de lenguajes construidos con este patrón: FPIC[(Kamin & Hyatt, 1997), Hawk [(Launchbury, Lewis, & Cook, 1999)], etc.

Compilador/ Intérprete extensible

Extender la implementación de un lenguaje existente puede también ser visto como una forma particular de lenguaje embebido. La diferencia usualmente es un asunto de grado. En un

enfoque de intérprete o compilador, la implementación sólo será extendida con algunas mejoras tales como nuevos tipos de datos u operadores. Para un enfoque embebido apropiado, las extensiones realizadas deben abarcar todas y cada una de las mejoras que requiere el lenguaje específico de dominio. No obstante, la extensión en ambos casos suele resultar dificultosa, aunque la inminente potencia con respecto a las operaciones que estas soluciones brindan, es motivo por el cual se suele optar por este enfoque. Un Ejemplo de DSL implementado de esta forma es DisTiL.[(Smaragdakis & Batory, 1997)]

Herramientas fuera de la plataforma comercial (COTS)

El enfoque de utilización de herramientas fuera de la plataforma comercial, suele involucrar la aplicación de funcionalidad existente en un método restringido, de acuerdo a las reglas de dominio. La gran cantidad actual de DSLs basados en XML es un ejemplo de este enfoque.[(Parigot & Courbis, 2005), (Gondow & Kawashima, 2002)]

En un DSL basado en XML, la gramática es descrita utilizando un DTD o un esquema XML (XML scheme) donde los nodos no-terminales son análogos a los elementos y los nodos terminales a los contenidos de datos. Las producciones pueden verse como definiciones de elementos donde el nombre de los elementos está del lado izquierdo y el modelo de contenido esta del lado derecho. El símbolo de inicio es análogo al elemento “document” en un DTD. De esta forma, utilizando un parser DOM o una herramienta SAX (Simple API for XML), el parseo se puede realizar fácilmente. Debido a que el árbol de parseo puede también ser un XML, se pueden utilizar transformaciones XSLT para la generación de código. Por lo tanto con herramientas de XML se puede implementar un compilador de un lenguaje de programación [(Germon, s. f.)].

A partir de aquí, y en los capítulos siguientes, se hace referencia al aporte real que busca hacer esta tesina al ámbito del control de calidad y los repositorios digitales, “la construcción y especificación de una herramienta para facilitar ciertas tareas recurrentes en la preservación y la curaduría”, siempre a partir de la utilización de ciertos recursos del ámbito de los lenguajes específicos de dominio. Con esto, se intenta asegurar la expresividad de la herramienta y su cercanía con los términos del dominio en cuestión.

Capítulo 5 | Especificación de la Herramienta

Introducción

Durante la etapa de análisis del problema y diseño de la solución, muchas de las decisiones tomadas fueron sustentadas sobre conceptos, patrones y herramientas del dominio de los Lenguajes Específicos de Dominio (capítulo 4). Este capítulo hace énfasis en, precisamente, la aplicación de estos lenguajes en el contexto de este trabajo, vinculándolos con los elementos del dominio de los repositorios digitales en general y con las características particulares del software DSpace sobre el que se construyó la herramienta.

En la primera parte de este capítulo se realiza una introducción a la herramienta y la especificación de la misma. A continuación, se describe la aplicabilidad de los distintos patrones, mecanismos y herramientas provenientes del ámbito de desarrollo de DSLs, que se utilizaron para proponer la especificación de una solución. Luego se analizan posibles implementaciones y se introduce (de manera muy resumida) la herramienta JSR-341 [(Srinivasan, 2006)], utilizada para facilitar la resolución de este trabajo. Finalmente, se concluye el capítulo definiendo el alcance de la implementación a realizar a fin de darle un enfoque pragmático al trabajo a partir de los objetivos planteados, siempre enmarcado dentro de un proyecto más extenso actualmente en desarrollo.

Descripción de la herramienta

Con las motivaciones y los objetivos planteados en el Capítulo 1 se intentó caracterizar un posible lenguaje específico de dominio, que pueda cumplir con los requerimientos de los objetivos. Para esto, se comenzó con una descripción sobre los atributos que serían deseables para una herramienta de este tipo. El objetivo aquí era poseer un mecanismo de consulta que permitiera a un administrador de un repositorio -o sea, un usuario con conocimientos avanzados en el dominio-, implementar distintos mecanismo de curaduría y control de calidad, con un cierto nivel de automatización.

La descripción del lenguaje es en sí mismo un resumen del aporte que se intenta hacer con esta investigación: la posibilidad de que exista una herramienta específica para el dominio de los repositorios con la suficiente expresividad y la facilidad de uso para resolver algunos de los problemas típicos de la gestión de repositorios digitales. En otras palabras, el objetivo de pensar primero en una descripción del lenguaje no solo fue establecer las bases para una solución al problema planteado anteriormente, sino también aportar en la formulación de un lenguaje con los conceptos específicos del dominio, para que la herramienta pueda implementarse con distintas metodologías, mecanismos y subdominios.

Un aspecto importante para proponer una posible solución es la definición y modelado de una *pseudo sintaxis abstracta*, algo que podría utilizarse como una *sintaxis abstracta* de un lenguaje en un futuro (ver trabajos futuros), pero que por lo pronto se abordará como forma de descripción de la herramienta que se quiere plantear, con el objetivo de ahondar en los requerimientos que la misma debería satisfacer. Con el objetivo de especificar una sintaxis que describa las operaciones pertinentes que una resolución con lenguajes de dominio debería considerar, se realizaron los siguientes pasos:

Análisis de las fuentes de conceptos

Se analizaron distintas fuentes desde donde extraer los conceptos claves en el dominio: las reuniones con expertos del dominio, la documentación del software DSpace (ver capítulo 3) y los casos de uso planteados fueron las fuentes principales.

Análisis de características deseables para un posible lenguaje

Se discutieron varios aspectos deseables para un lenguaje de acuerdo a los mecanismos de control antes mencionados. Ya que este lenguaje tenía que permitir implementar tareas de curaduría en un repositorio gestionado con DSpace, se plantearon las siguientes características:

- **Simplicidad:** el lenguaje debería poseer un equilibrio entre legibilidad y facilidad de escritura: demasiada simplicidad conlleva una disminución en la legibilidad y viceversa. Sin embargo, la simplicidad es una característica deseable puesto que aumenta la confiabilidad del software (en este caso de las consultas) construído con el lenguaje ya que al ser más sencillo, la verificación y detección de errores es más sencilla.
- **Tipos y estructuras de datos:** la idea era que el lenguaje permitiera organizar la información de acuerdo a su tipo y en estructuras de datos convenientes. Esto era deseable porque los tipos y estructuras de datos aumentan la confiabilidad ya que es posible el chequeo de tipos. No obstante, para el lenguaje en el que se estaba pensando, los tipos y las estructuras de datos iban a estar definidos por el contexto en el cual se ejecutaba: Dspace (descrito en capítulo 3).
- **Diseño de sintaxis:** los símbolos y elementos a utilizar y la forma de combinarlos, eran un tópico clave en la selección del lenguaje a utilizar y la especificación de éste. El diseño de la sintaxis influye en la legibilidad y facilidad de escritura, en la confiabilidad y en los costos y además posee una importancia clara a la hora evaluar la facilidad de aprendizaje del lenguaje: una sintaxis amigable para el usuario, con símbolos familiares y herramientas pedagógicas como conceptos de anclaje.[(Moreira, 1997)]
- **Soporte para abstracción:** que el lenguaje de consulta pueda abstraer ciertos conceptos de repositorios para minimizar la complejidad de los problemas a resolver agrupándolos de acuerdo a ciertas características, era algo deseable. Se puede comprobar que esta propiedad aumenta la legibilidad y facilidad de escritura así como la confiabilidad.
- **Expresividad:** un lenguaje expresivo gozará de una mayor naturalidad para expresar sus sentencias e incluso una mayor potencia en relación a las operaciones que podrá representar. No obstante, si bien un lenguaje muy expresivo tendrá una gran legibilidad y confiabilidad, otras propiedades como la facilidad de escritura y el costo de aprendizaje se verán disminuidas. [(Ghezzi & Jazayeri, 1986)]

Identificación de conceptos

Para identificar conceptos importantes del dominio que deberían estar presentes en el lenguaje, se utilizaron varias fuentes y se contemplaron diferentes reglas de validación sobre los conceptos encontrados. La mayoría de estos conceptos surgieron del agrupamiento de términos conocidos y utilizados en el ámbito de repositorios; otros aparecieron del mapeo directo con el problema y con los casos de usos planteados.

Como resultado de esta identificación, se tuvieron en cuenta aquellos conceptos que disponían de una semántica y se limitó el dominio a los estrictamente necesarios para resolver los casos de uso planteados (Par más información, Ver *Capítulo 3, sección "Modelo de DSpace"*).

La lista de conceptos utilizados quedó de la siguiente forma: ITEM - COLECCION- COMUNIDAD- METADATO- BUNDLE - BITSTREAM- SITE-CONTEXT

Requerimientos funcionales

El lenguaje a utilizar debía, ante todo, poder satisfacer los objetivos planteados y permitir distintos casos de uso, necesarios para realizar las tareas de control de calidad. Para establecer los requerimientos funcionales del lenguaje, los casos de uso se simplificaron en las operaciones más generales, que el mismo debía proveer para cumplirlos. Como resultado de esto, surgieron las siguientes operaciones a implementar:

Operación 1: Validación de condiciones

El lenguaje debe dar soporte para validar si determinados elementos cumplen o no con una condición especificada. Dicho de otro modo, el lenguaje debe permitir aplicar una regla determinada que, dado un objeto particular, valide una condición, donde una condición es una operación entre una o muchas subexpresiones que devuelven un valor de verdad.

Formalmente, la validación puede definirse como: dado un objeto O y una regla R que define una o más condiciones C_1 C_2..C_n, se aplicará R sobre O, evaluando si O cumple C_i para todo i = 1..n o para al menos uno, según sea una conjunción o disyunción respectivamente.

De la definición anterior se observa que la operación de proyección de un cierto subconjunto de objetos es, en realidad, una validación que consiste en una regla R de condición única C_1. Dicha condición C_1 debe validar la pertenencia del objeto evaluado al subconjunto que se esta proyectando. Esto puede esclarecerse con el siguiente ejemplo:

Dado un objeto cualquiera O, se evalúa la regla R que sólo valida la condición C_1 siguiente: "que el objeto O sea del tipo ITEM". Puede verse claramente que esto es una operación de proyección de un ITEM sobre un conjunto de objetos cualquiera. Si esta misma validación se aplica sobre un conjunto {O_1,O_2,..O_n} de objetos, se estarían proyectando todos los objetos pertenecientes al subconjunto "ITEMS" y si esta validación se utiliza en una operación de Selección, entonces se puede obtener todos los objetos O_1,O_2,..O_n que cumplan con la validación y por lo tanto sean ITEMS.

Existen varios ejemplos de usos posibles para esta operación, algunos de estos son:

- Verificar que los artículos posean: algún autor, fecha o título
- Validar si los BUNDLE de un ÍTEM determinado poseen o no al menos un BITSTREAM
- Validar si un Objeto es un ITEM, COLECCIÓN o COMUNIDAD
- Verificar existencia de un metadato obligatorio en un OBJETO

Operación 2: Selección de objetos

El lenguaje debe permitir seleccionar ciertos objetos que cumplan con una o varias condiciones determinadas, donde cada una de estas condiciones sea, en realidad, una *validación,* como las mencionadas en la operación anterior. Esto resultaría muy útil, puesto que los casos de uso planteados para el lenguaje no requieren que el usuario conozca todo el

modelo de datos y, por lo tanto, el lenguaje no involucra necesariamente, que el usuario conozca todo el dominio de Dspace.

Definida formalmente, la operación de selección permite seleccionar un subconjunto de tuplas de una relación (R) que cumplan la(s) condición(es) P.

En este caso, la operación de selección debe permitir al usuario, escoger el dominio de aplicación de su consulta. Es decir, los elementos del repositorio sobre los cuales se va a aplicar la consulta construida.

La selección es importante en cualquier lenguaje de consulta, porque define el alcance del mismo: las operaciones que el lenguaje puede aplicar, se podrán ejecutar sobre aquellos elementos que se puedan referenciar sobre el mismo.

Si se toma como ejemplo la herramienta de validación a diseñar, el análisis de los conceptos y de las fuentes arrojó que se debía contemplar la representación de todos los elementos perteneciente al modelo de datos de DSpace (Ver Capítulo 3, sección “Modelo de DSpace”). De esta forma, el usuario podría referenciar los elementos contenidos en el modelo de un repositorio, y aplicar consultas sobre los datos que se mantienen sobre los mismos.

Los siguientes son ejemplos de operaciones que se pueden hacer con la selección:

- Mostrar ITEMS de una COLECCION o COMUNIDAD que cumplan con una validación
- Listar OBJETOS cuyo valor no posea un ID de authority válido
- Proyectar todas las OBRAS de un autor determinado
- Proyectar todos los LIBROS que no poseen un ISBN

Operación 3: Transformaciones

De manera incremental, apareció la idea de que el lenguaje podría proveer mecanismos para realizar modificaciones sobre los objetos seleccionados. La transformación puede verse como la operación de actualización típica sobre los objetos a los que se les aplicó una *selección, como la mencionada en la operación 2*.

Formalmente, la operación puede describirse así: dado un objeto O seleccionado (que cumple una regla R determinada por medio de la selección), aplicar una función de transformación T(O), que lleva al objeto O a un objeto O’ a partir del cambio de al menos uno de los metadatos M_1, M_2,..,M_n que pertenecen a O.

Dicho de otro modo, permitir que a partir de un conjunto de objetos seleccionados debido a si cumplen con una condición determinada (validación) o no, se puedan aplicar modificaciones sobre un atributo determinado, siempre a partir de la alteración de uno o varios metadatos específicos.

Para esclarecer la operación de transformación, se pensaron distintos casos concretos de uso, algunos ejemplos son:

- Añadir una licencia de uso a los ITEMS de una COMUNIDAD
- Eliminar un METADATO indeseado de un OBJETO específico
- Cambiar el valor de un METADATO de un conjunto de OBJETOS
- Eliminar un BITSTREAM de un BUNDLE de un ITEM específico

Representación de la Herramienta

Como ya se dijo (capítulo 4), para la representación de la herramienta se utilizaron diversos mecanismos propios del ámbito de los DSL. Esto también formaliza los atributos que las posibles implementaciones deberían poseer. [(Andino & Ruiz, 2009)]

Construcción de árboles AST

Con las operaciones básicas definidas en la “Descripción de la Herramienta”, se plantearon y construyeron los distintos árboles de sintaxis abstracta, para poder identificar los términos clave en el dominio y las construcciones más básicas que se podrían armar con las sentencias escritas en el lenguaje.

Para definir los AST, se pensó en los casos de uso antes enunciados y en brindar herramientas a los usuarios para que puedan ejecutarlos, con la mayor simplicidad posible.

AST para validación

El primer AST en el que se pensó fue el correspondiente a la operación de validación. Por estar involucradas en las demás operaciones, la operación de validación puede verse como el sub-árbol más próximo a los nodos hoja de un árbol mayor, que posee todas las construcciones del lenguaje.

Como su especificación lo define, una validación es una expresión que devuelve un valor de verdad sobre alguna característica específica de algún elemento del repositorio. Para definirla formalmente, se puede ver a una validación como la construcción siguiente[1]:

VALIDACIÓN::= PREDICADO_SIMPLE | PREDICADO_SIMPLE OPB PREDICADO_SIMPLE

PREDICADO_SIMPLE::= EXPRESIÓN_BOOLEANA_UNARIA | EXPRESIÓN_BOOLEANA_BINARIA

EXPRESIÓN_BINARIA_UNARIA::= ‘NOT’ EXPRESIÓN

EXPRESIÓN_BOOLEANA_BINARIA::= EXPRESIÓN OPB EXPRESIÓN

EXPRESIÓN::= EXPRESIÓN_COMPARACION | PREDICADO_SIMPLE

EXPRESIÓN_COMPARACION::= PROYECCIÓN_DSPACE ‘=’ VALOR

VALOR:= PROYECCIÓN_DSPACE | LITERAL

OPB::= AND | OR | XOR

PROYECCIÓN_DSPACE::= DSPACE_ATRIBUTO| METADATA

DSPACE_ATRIBUTO ::= DSPACE_OBJECT ‘.’ STRING_LITERAL

DSPACE_OBJECT ::= ITEM | COLECCIÓN | COMUNIDAD | SITIO | CONTEXTO |BUNDLE | BITSTREAM

Con esta definición, se puede formular el árbol AST que se puede visualizar en la *figura 5.1* de ejemplo. ejemplificar la descripción de la operación):

1 Se muestran los términos en una notación similar a EBNF, solo con un fin demostrativo para que el lector entienda las características buscadas en el lenguaje. Esta descripción no busca ser un planteo sintáctico para un lenguaje real y mucho menos debe ser considerada como tal.

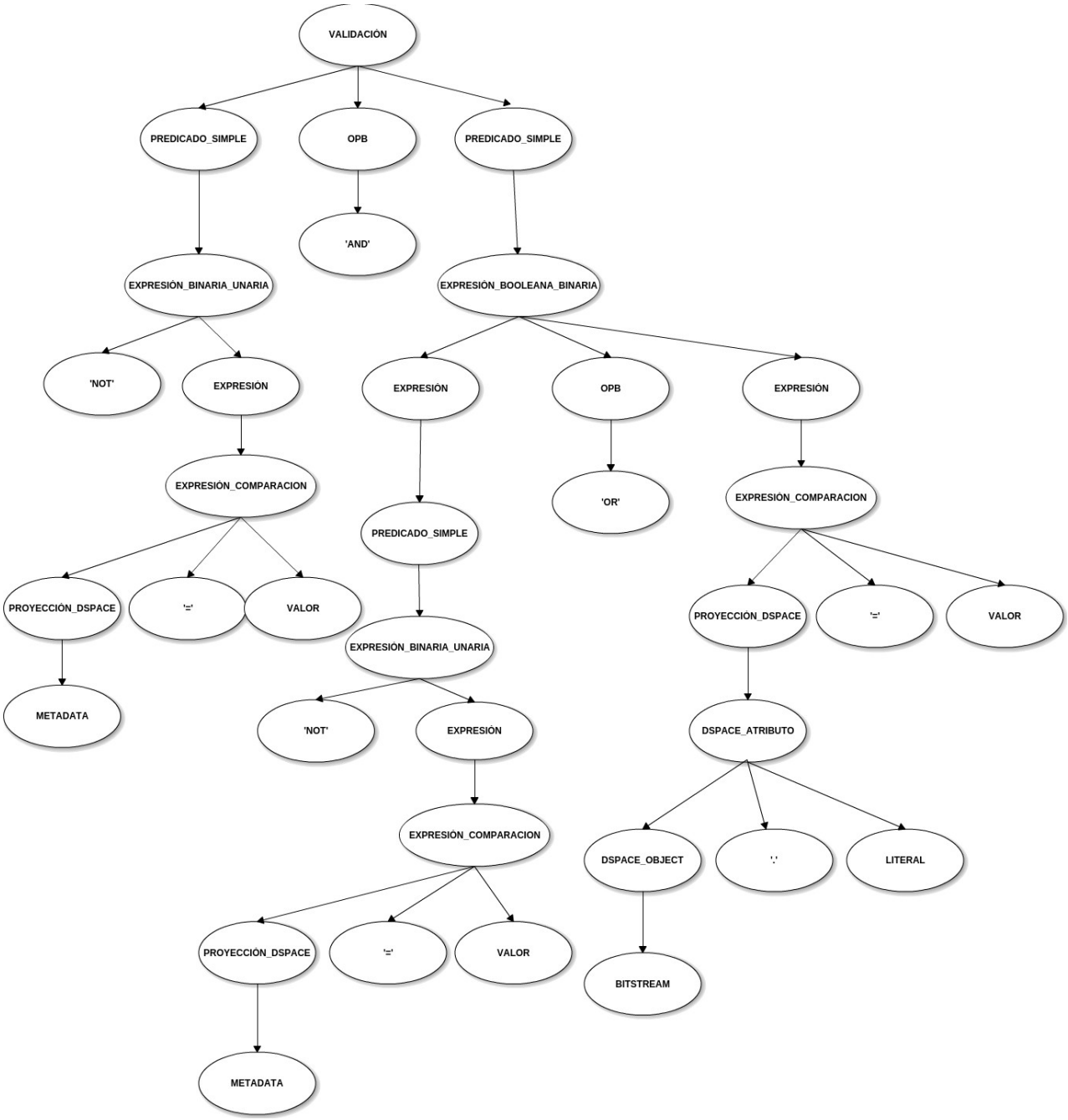

Figura 5.1: uno de los posibles AST que se pueden construir, cuya única finalidad es ejemplificar la pseudo construcción planteada.

De aquí, se puede observar que las hojas del árbol son términos bien definidos en el dominio específico tratado, por lo que son considerados nodos terminales debido a que hacen referencia directa al modelo semántico que se busca plantear más adelante, y que servirá de estructura principal para definir la expresividad que debe considerar la solución a implementar. Así como también, a lo visto en el modelo de DSpace, planteado en el capítulo 3.

AST para la selección

Tal y como se ha definido anteriormente, la operación de selección representa el acto de elegir o reportar aquellos elementos del modelo que cumplen con una validación específica, de acuerdo a la estructura definida previamente. Por lo tanto, una construcción de ejemplo posible para la operación de selección sería la que se brinda a continuación.

SELECCIÓN::= PROYECCIÓN_DSPACE ‘SELECT’ VALIDACIONES

VALIDACIONES::= VALIDACIÓN_UNARIA | VALIDACIÓN_UNARIA OPB VALIDACIONES

VALIDACIONES_UNARIA::= VALIDACIÓN | ‘NOT’ VALIDACIÓN

Nótese que se reutilizan *términos no terminales* de la validación y la validación como tal, intentando definir con formalidad, que una selección, también puede involucrar una sucesión de validaciones. En la *figura 5.2* se muestra un árbol AST de ejemplo.

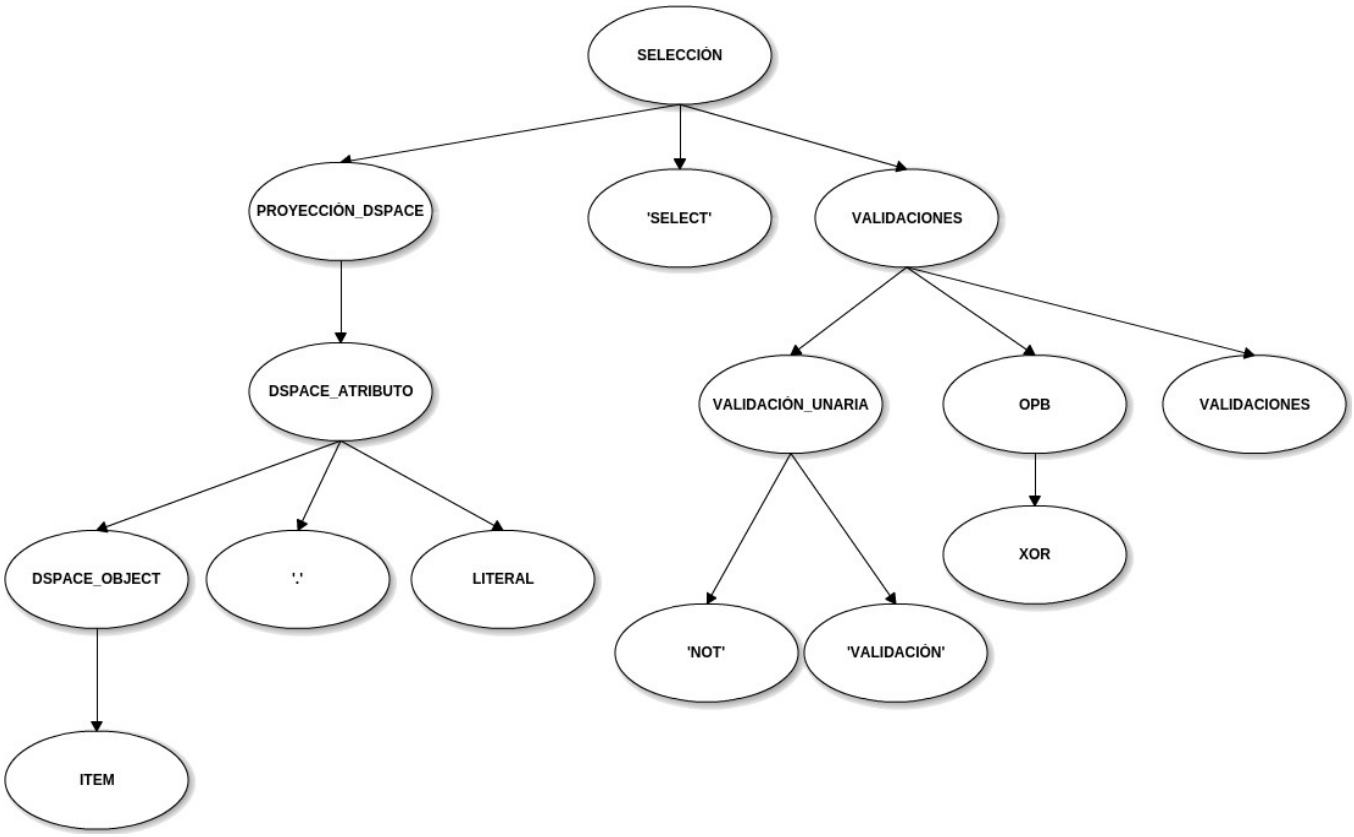

Figura 5.2: un posible árbol AST para la selección, involucra a la operación de validación definida anteriormente

AST para la transformación

La operación de transformación consiste en aplicar una función de modificación al resultado de una selección válida. Las transformaciones a aplicar serán equivalentes a todo el conjunto aislado como producto de la selección y no existe aplicación parcial, es decir, se aplicarán sobre todos los elementos del conjunto o sobre ninguno. Las funciones de

transformación planteadas son tres: modificación de atributo y/o metadato (UPDATE), adhesión de atributo (ADD) y eliminación de atributo no requerido (ERASE). En el lenguaje, estarán representadas por los respectivos operadores de transformación. De este modo, la transformación podría definirse de la siguiente forma:

TRANSFORMACIÓN::= SELECCIÓN OPTU VALOR_DE_TRANSFORMACIÓN

VALOR_DE_TRANSFORMACIÓN::= TRANSFORMACIÓN_BINARIA | TRANSFORMACIÓN_UNARIA

TRANSFORMACIÓN_BINARIA::= TRANSFORMACIÓN_UNARIA '+' TRANSFORMACIÓN_BINARIA

TRANSFORMACIÓN_UNARIA::= METADATO ':=' LITERAL

OPTU::= UPDATE | ADD | ERASE

Notar que se formaliza la situación de que una transformación involucra una selección, para definir los elementos sobre los cuales operar. Además, las operaciones de transformación pueden estar sujetas a cambios, si se encuentran nuevas necesidades o casos de uso que se puedan abarcar. En la *figura 5.3*, se puede ver uno de los AST probables, para ejemplificar la operación.

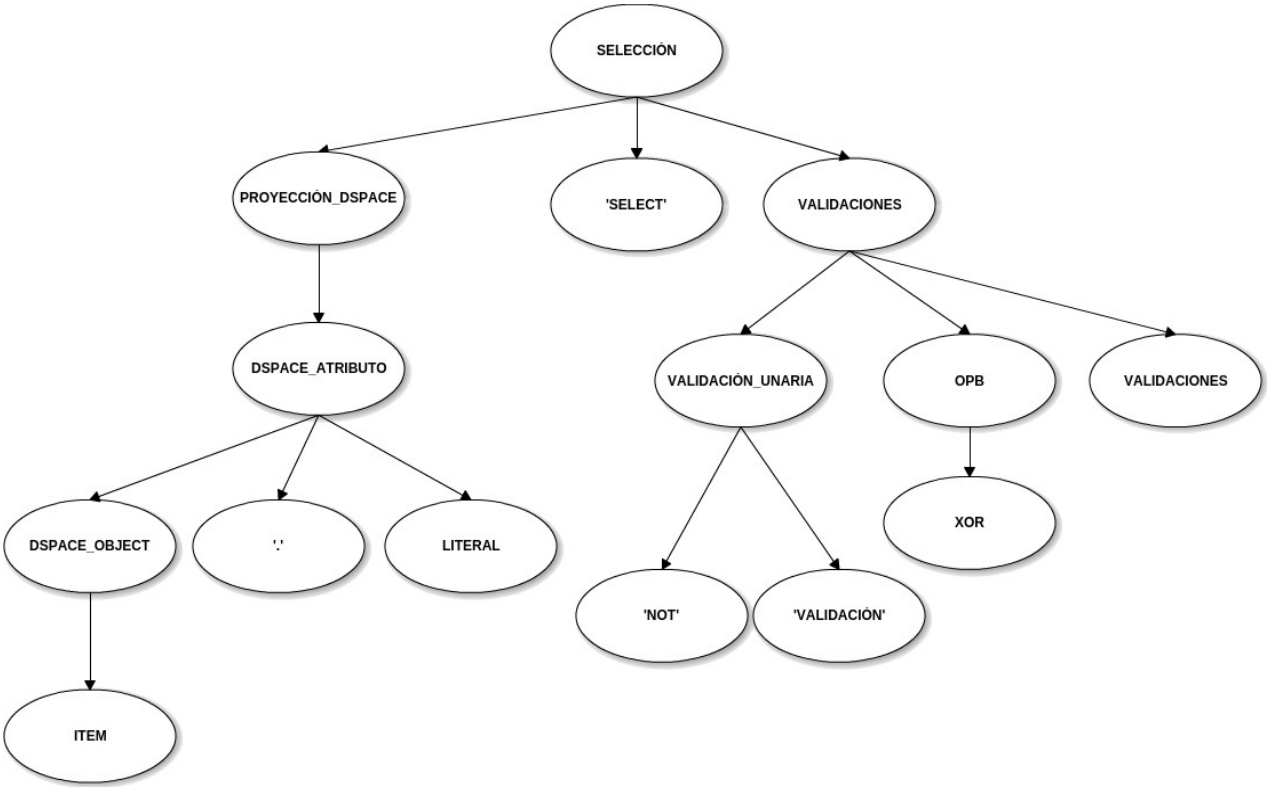

Figura 5.3: un posible árbol AST para la operación de transformación planteada. Se puede ver que la operación de transformación involucra una selección de elementos.

Construcción del modelo semántico

A partir de los términos identificados durante la definición de los AST se aplicó el patrón de diseño de "*Modelo Semántico*", utilizado en el ámbito de los DSL [(Fowler, 2010)]. El objetivo

principal fue la definición de un modelo que reúna los conceptos del dominio que debería contemplar una solución. De modo que a partir de este modelo, se puedan realizar posibles implementaciones, donde las mismas se adapten a cada problema específico que se quiera resolver. De esta forma, un aporte concreto de este trabajo radica en la caracterización de una herramienta específica en el dominio de repositorios y su especificación reutilizable e incremental, que puede ser efectuada con distintos mecanismos y con diferentes objetivos.

El modelo semántico permite definir una capa de abstracción sobre la implementación de la herramienta y así aislar por un lado los términos del dominio, y por el otro la implementación, principalmente porque muchos patrones de implementación permiten construir una solución pero pueden interferir en la gramática de la misma. En cambio, si se toma alguna herramienta de este tipo, la implementación podría ser poco representativa del dominio en cuestión (usar términos y operaciones poco familiares), algo que no es para nada deseable.

Por lo dicho en el párrafo anterior, un paso muy importante en el planteo de una solución fue construir un modelo semántico mediante la definición de una especie de *sintaxis abstracta común* que cualquier implementación deberá respetar. De este modo, la gramática de la solución concreta obtenida a partir de una implementación posible puede variar, pero conservará el objetivo especificado para el DSL en cuestión.

El modelo formado a partir de los nodos hoja de los AST (aquellos nodos terminales que adquieren significado e importancia en el dominio de DSpace) y las operaciones que cada árbol representa, se ve en la *figura 5.4.*

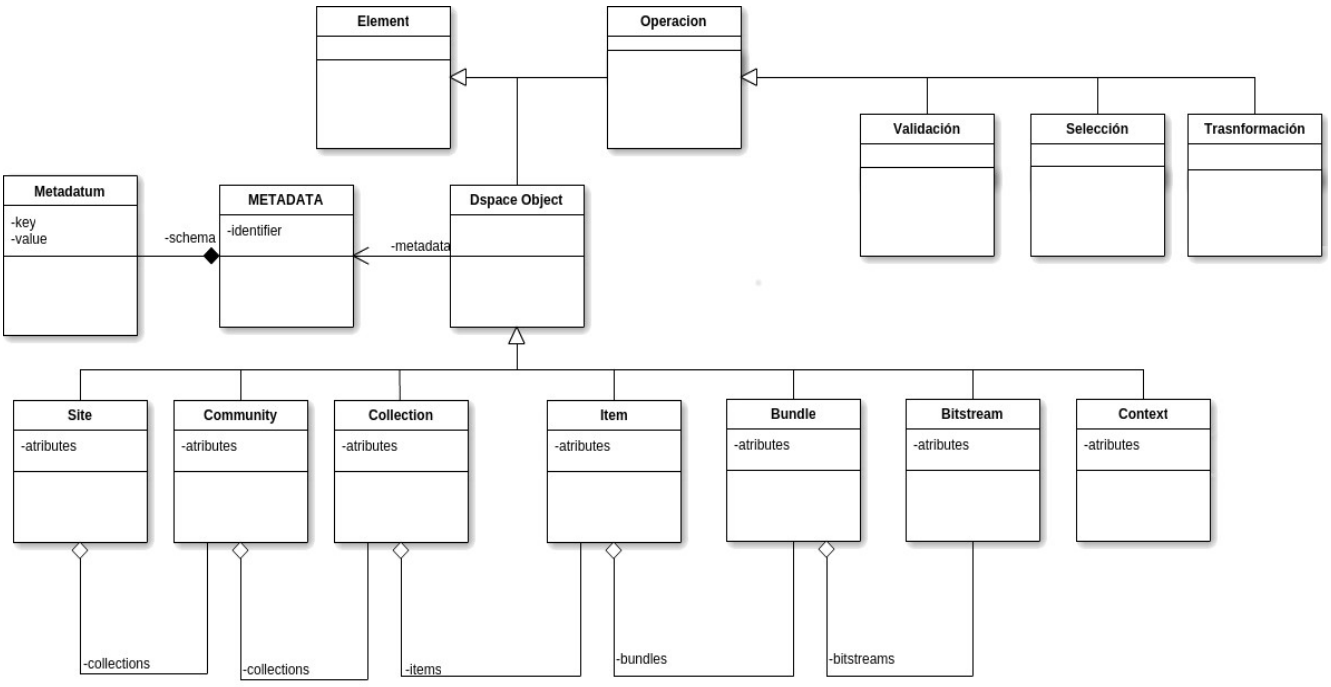

Figura 5.4: El metamodelo resultante para la herramienta, que agrupa los términos y operaciones que deberían tomarse en cuenta para plantear una solución.

Selección de Herramienta para Implementación

A partir de una visión global de los distintos mecanismos para implementar el desarrollo de la solución planteada *(Ver Cap 4)*, se buscaron herramientas que faciliten la implementación y la encuadren dentro de alguno de los enfoques anteriores. Si privilegiaron herramientas que

permitan desarrollar lenguajes específicos de dominio internos a partir de la implementación de preprocesamiento, lenguajes embebidos o COTS. Los otros enfoques se descartaron debido a que su implementación era demasiado costosa, por lo que podría requerir un tiempo y esfuerzo que superen el alcance de este trabajo. No obstante, como se ha apuntado a separar la implementación de la definición del lenguaje, la implementación de un compilador o intérprete para el lenguaje de control de calidad, puede ser tema de trabajo futuro. *(Ver sección "Trabajos Futuros", Capítulo 8)*

En este contexto, se consideraron algunas alternativas que serán comentadas a continuación.

Java Fluent API

La primer alternativa de implementación evaluada, fue la realización de una fluent API o interfaz fluida, en el contexto de un lenguaje embebido.-Se denomina como interfaz fluida a un patrón de implementación para implementar lenguajes específicos de dominio internos [(Fowler, 2010), (Mernik et al., 2005; Voelter, 2013)]. Este patrón, puede aplicarse en prácticamente cualquier lenguaje, sin embargo aquí se evaluó la alternativa de implementación en JAVA, por ser el lenguaje en el que esta desarrollado el software DSpace y en donde se tienen los casos de usos planteados para la herramienta de validación.

Una fluent API es simplemente una API que ha sido desarrollada de tal forma que sus métodos puedan ser utilizados para construir programas o scripts con la misma. Esto significa que se puede implementar un DSL interno utilizando los métodos de la fluent API para representar los términos propios del mismo. Para realizar esto, existen una serie de patrones de diseño, aplicados en el diseño de las interfaces de la API, que permiten distinguir la brecha entre una API normal y una fluida. [(Fowler, 2010)]

A primera vista esta metodología parecía viable, debido a su relativa facilidad de implementación y a que el desarrollo de un módulo de expresiones que entienda el lenguaje planteado podría ser incorporado al contexto de Dspace de forma relativamente sencilla. Sin embargo, si bien cualquier DSL interno está fuertemente acotado por las restricciones del lenguaje anfitrión, en el caso de las interfaces fluidas esto parecía profundizarse. La mayoría de los patrones de diseño utilizados para implementar una fluent API en JAVA [(Sanaulla, 2013)] (encadenamiento de métodos, secuencia de funciones, funciones mezcladas) no sólo influyen en la implementación, sino que restringen la gramática del lenguaje. En muchos casos este no es un problema, debido a que un DSL interno siempre tendrá los términos de su lenguaje anfitrión (JAVA) incorporados. No obstante, para el caso específico de este lenguaje, es de suma importancia mantener la gramática tanto como sea posible, debido a que se se ha construído privilegiando ciertas características importantes como simplicidad y ortogonalidad, con el objetivo de mejorar su facilidad de aprendizaje.

Especificación del lenguaje de expresión JSR-341

Una de las herramientas estudiadas fue la especificación para lenguajes de expresión (JSR-341) e implementaciones de la misma [(Chung, 2013)]. El objetivo era reutilizar componentes determinados allí, adaptandolos al modelo semántico del lenguaje y obtener un lenguaje de expresión, de una gramática similar a la planteada, que pueda resolver total o parcialmente los casos de uso planteados en capítulos anteriores. No obstante, el desafío para llevar a cabo esta implementación, era establecer un equilibrio entre la expresividad que proponía la librería (que se puede ver en el Capítulo 6), y la que se necesitaba realmente en en la herramienta a desarrollar. Tal y como lo muestra la *figura 5.5*, el usuario debía tener que

conocer el metamodelo de la herramienta, el modelo de DSpace(ver capítulo 3) y la sintaxis concreta planteada por la librería, para poder utilizar la herramienta.

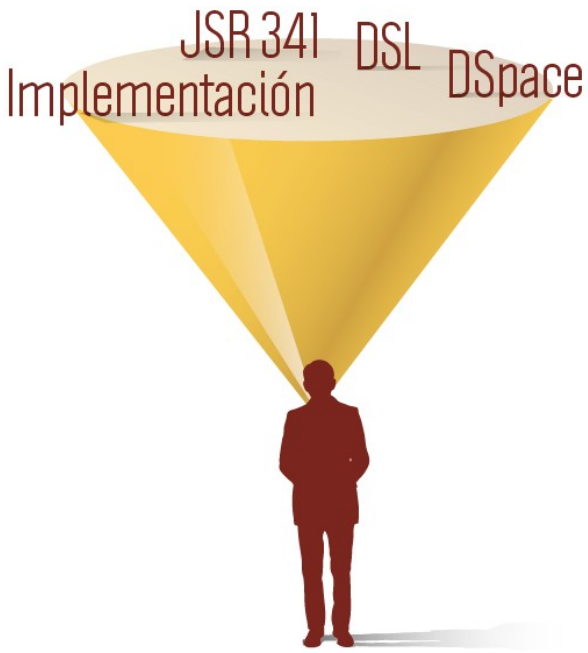

Figura 5.5: Conocimientos que se deberían nuclear para poder ejecutar la herramienta. Implementación+DSL podrían resumirse si el usuario conociera el metamodelo utilizado.

En este contexto, se planteó una implementación que permita acotar la expresividad, para facilitar la utilización. Se tomaron como referencia algunos ejemplos de uso de esta especificación: OGNL [(Apache Commons, 2013)], MVEL [(Codehouse, 2009)], JBoss EL[(Seamframework, 2009)].

Los resultados y las decisiones sobre la implementación de la herramienta, se continúan tratando en el capítulo 6.

Alcance de la implementación

Con la caracterización de la solución definida, y la selección de una posible herramienta para implementar esta solución, se procedió a implementar una resolución considerando los conocimientos que se tenían y buscando un desarrollo en un plazo corto, que pueda cumplir con los objetivos planteados para este trabajo y que resuelva las necesidades vigentes en el repositorio institucional.

Por lo dicho en el párrafo anterior, se buscó un lenguaje que apunte principalmente a disminuir el costo de aprendizaje, sin perder demasiada expresividad. Para esto, se debe buscar la expresividad a partir de la utilización de conceptos del marco de los repositorios y la facilidad de aprendizaje surgirá de la familiaridad que presentan estos conceptos para los usuarios a los que está destinada la implementación. Esto último se condice también con la selección de la herramienta, puesto que la librería escogida para facilitar la implementación, se caracteriza por buscar objetivos similares a la herramienta que se quiere desarrollar.

Capítulo 6 | Implementación del Lenguaje: Módulo de expresiones

Análisis

Con una especificación de la herramienta ya realizada y con el objetivo de efectuar una posible implementación que sirva como ejemplo, se desarrolló un módulo de expresiones para el contexto de DSpace 5, utilizando la especificación mencionada en el capítulo 5, definida en JSR-341 [(Chung, 2013)]. La idea del análisis es mostrar las ventajas y todas las posibilidades que se presentan al disponer de una herramienta de este tipo en el ámbito de los repositorios.

En este contexto, se buscó responder a la necesidad que se planteó con anterioridad tanto en los trabajos previos como en la motivación de esta tesina: el desarrollo de una herramienta para control semi-automático de calidad en repositorios digitales.

Al implementar un DSL interno y, en este caso, utilizar una herramienta que facilita ciertos aspectos de la implementación, resulta necesario adecuar la sintaxis planteada para el lenguaje, a lo que se puede implementar en función de la metodología utilizada. Por esto, la especificación del módulo de expresiones busca realizar un análisis de lo que se va a intentar resolver con la implementación. Se definirá una alcance de implementación que sea pensado en base a lo que ya se planteó: las operaciones posibles para el lenguaje definido previamente.

La etapa de análisis en la implementación consiste en el estudio del aporte real que representa la JSR-341 para el desarrollo, sus características principales, la sintaxis que propone y su forma de utilizarla. Luego, se describe la forma de adecuarla al problema para el que se requiere una solución. Finalmente, se definen los pasos a seguir para utilizar la herramienta, y se procede con el diseño de una solución.

Especificación JSR-341

Según la documentación [(Chung, 2013)], la JSR-341 es una especificación para la tercera versión del lenguajes de expresiones (EL 3.0), desarrollado por un grupo de expertos en el marco del proceso de la comunidad Java [(Java Community Process, s. f.)].

El lenguaje de expresiones descrito en esta especificación (cuya implementación se puede encontrar en los paquetes oficiales javax.el y javax.el-api, [(Process, s. f.)]), fue diseñado originalmente como un lenguaje simple, acotado a los requerimientos de la capa de presentación de las aplicaciones web (para controlar la forma en cómo las aplicaciones web muestran los datos) . Por esto, provee:

- Una sintaxis concreta simple, restringida a la evaluación de expresiones
- Un mecanismo para definir variables y propiedades anidadas
- Operadores relacionales, lógicos, aritméticos, condicionales y de vacío (empty)
- Posibilidad de implementar funciones propias, como métodos estáticos de las clase JAVA del modelo utilizado
- Una leve comprobación semántica, realizada a partir de valores por defecto y chequeos y conversiones de tipos nativos de Java. [(«Primitive Data Types», s. f.)]
- Herramientas para utilizar el EL como una herramienta autónoma, a través de una API de evaluación directa y para manipular el entorno de evaluación del EL (se explicará más adelante)

Para poder utilizar y adaptar e EL especificado, es necesario conocer la sintaxis concreta que propone, a continuación se realiza de lo más representativo de esta sintaxis, que se puede encontrar completa en la documentación de la especificación, ya referenciada. [(Chung, 2013)]

Sintaxis concreta del EL

Como ya se dijo, el EL provee una sintaxis concreta bastante simple. Los objetos del modelo son accedidos por nombre. Se hace foco en un operador de **'.'**, que se puede utilizar para acceder a las propiedades, siempre que el nombre de la propiedad siga las convenciones de los identificadores de Java [(Oracle, 1999)]. Además el operador se puede utilizar para invocar métodos de los objetos referenciados, siempre que los mismos tengan el modificador de acceso *'public'*.

Por otro lado, las comparaciones relacionales utilizan los operadores relacionales estándares para Java [(Oracle, s. f.-d)]. Las comparaciones pueden realizarse contra otros valores, o contra literales booleanos, cadenas, enteros o punto flotante.

Los operadores aritméticos pueden utilizarse para computar valores enteros o de punto flotante. Existen operadores binarios para suma (+), resta (-), multiplicación (*), división (/,div) y resto o módulo (%,mod). A esto, se le añade un operador unario de negación (-).

También están disponibles los operadores lógicos, tanto los binarios 'y' (&&, and), 'o' (or, ||), como el unario 'no' (!, not).

Además, existen operadores para concatenar dos cadenas (+=), que asocian los valores de dos cadenas y retornan una nueva cadena con los valores unidos.

Se añade un operador prefijo que se utiliza para determinar si un valor es nulo o vacío (empty; y el operador condicional (A?B : C) que evalúa B o C, de acuerdo al resultado de la evaluación de A. En adición a esto, existe el operador de asignación (=) que asigna un valor determinado.

La procedencia de los operadores queda definida de la siguiente forma forma, desde el de mayor prioridad al de menor (de izquierda a derecha):

- [ ] .
- ( )
- - (unario) not ! empty
- * / div % mod
- + - (binario)
- +=
- < > <= >= lt gt le ge
- == != eq ne
- && and
- || or
- ? :
- -> (Expresión Lambda)
- =
- ;

Existen también distintos operadores de colecciones, que permiten manipular y procesar colecciones nativas de Java, tales como: conjuntos, listas y mapas [(«java.util (Java Platform SE 7)», s. f.)]. Estos operadores se utilizan sobre un objeto del EL, que representa un *stream* de los elementos de la colección. Por lo tanto, para aplicar la operación a una colección (también se puede hacer sobre un arreglo Java [(Oracle, s. f.-a)]), se debe recurrir al método

stream() del lenguaje. Con un stream de objetos, las operaciones más comunes que se pueden hacer son: (muchas se utilizan en las expresiones de ejemplo de más adelante)

- *Stream<S>.filter((S->boolean) predicado)*: produce un nuevo stream o colección de los objetos en donde el predicado que se pasó como parámetro es verdadero. Es decir, filtra objetos por una condición determinada.
- *Stream<S>.forEach((S->void) consumer):* este método invoca a la función que como parámetro en 'consumer', para cada elemento en el stream(). Es decir, permite aplicar una función a todos los elementos de la colección.
- *Stream<S>.anyMatch((S->boolean) predicado)*: retorna verdadero si algún elemento del stream satisface la condición que llega en el predicado. Es decir, comprueba si algún elemento de la colección cumple con la condición especificada.
- *Stream<s>.sorted(((p,q)->int) comparador):* produce un nuevo stream, que contiene a los elementos del stream ordenados con el orden impuesto por el comparador, aunque si no hay comparador, utiliza el orden natural.

Con una descripción mínima de la sintaxis concreta del lenguaje de expresiones, se pasó a analizar su forma de utilización y, posteriormente, a adaptarlo para el dominio del problema planteado.

Utilización del lenguaje de expresiones

El lenguaje de expresiones debe ser extendido con el comportamiento correspondiente al modelo propio, a través de una API[2], que permite la definición de una serie de objetos para construir un lenguaje especializado. Esto presenta varias ventajas para los desarrolladores del lenguaje, como por ejemplo:

- contar con una sintaxis sencilla, restringida a la evaluación de expresiones,
- la posibilidad de uso de funciones y variables,
- contar con operadores relacionales, lógicos, aritméticos, condicionales y de chequeo de vacíos de manera nativa,
- disponer de un control semántico sobre tipos y un parser sintáctico basado en ValueExpressions de JAVA. [(«Value and Method Expressions », s. f.)]

El modelo de clases del lenguaje de expresiones presenta una clase instanciable que contiene toda la API necesaria para poder manipular un modelo propio con el lenguaje, de forma similar al patrón de diseño *Fachada* [(Gamma, Helm, Johnson, & Vlissides, 1995)], solo que con un objeto instanciable, con estado interno. Esta clase, conocida como ELProcesor, tiene asociados dos objetos que brindan soporte a la interpretación de las expresiones que se pueden validar con el lenguaje. El primero es una instancia de ElManager, donde se definen todas las consideraciones del contexto del lenguaje, incluso los mecanismos para alojar variables o funciones estáticas. El otro, es una instancia de una clase factory que viene por defecto en el paquete *javax.el*, en este caso la implementada para *Glassfish EL* [(«GC: ExpressionFactoryImpl », s. f.)](una implementación del lenguaje regular, para utilizar un servidor Glassfish) [(Manual, 2013)]. Esta última se utiliza para evaluar el string como expresión, parsearlo e interpretarlo en el contexto que se definió con el *ELManager*.

2 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.

Como se mencionó en el párrafo anterior el *ELManager* mantiene la información relativa al contexto en donde se evaluará una expresión del lenguaje. Para hacer esto, permite definir una instancia propia o estándar de un *ELContext*. En este caso, se utilizó una instancia estándar que está compuesta por un mapeador de funciones por defecto (permite que se definan funciones estáticas que luego serán invocadas dentro del lenguaje), un mapeador de variables por defecto (permite definir variables que luego pueden ser referenciadas desde el lenguaje) y una serie de *ELResolvers*, encargados de resolver la evaluación de una expresión en el lenguaje a partir de la interpretación de un modelo determinado. Para este caso, se utilizaron los resolvedores estándares y se añadió un *DSpaceResolver*, que podría dar soporte a funciones específicas del contexto, de ser necesarias más adelante. En la figura 6.1 puede verse un diagrama aproximado (en estandarización UML[(Omg, 2011)]) del modelo que utiliza el EL.

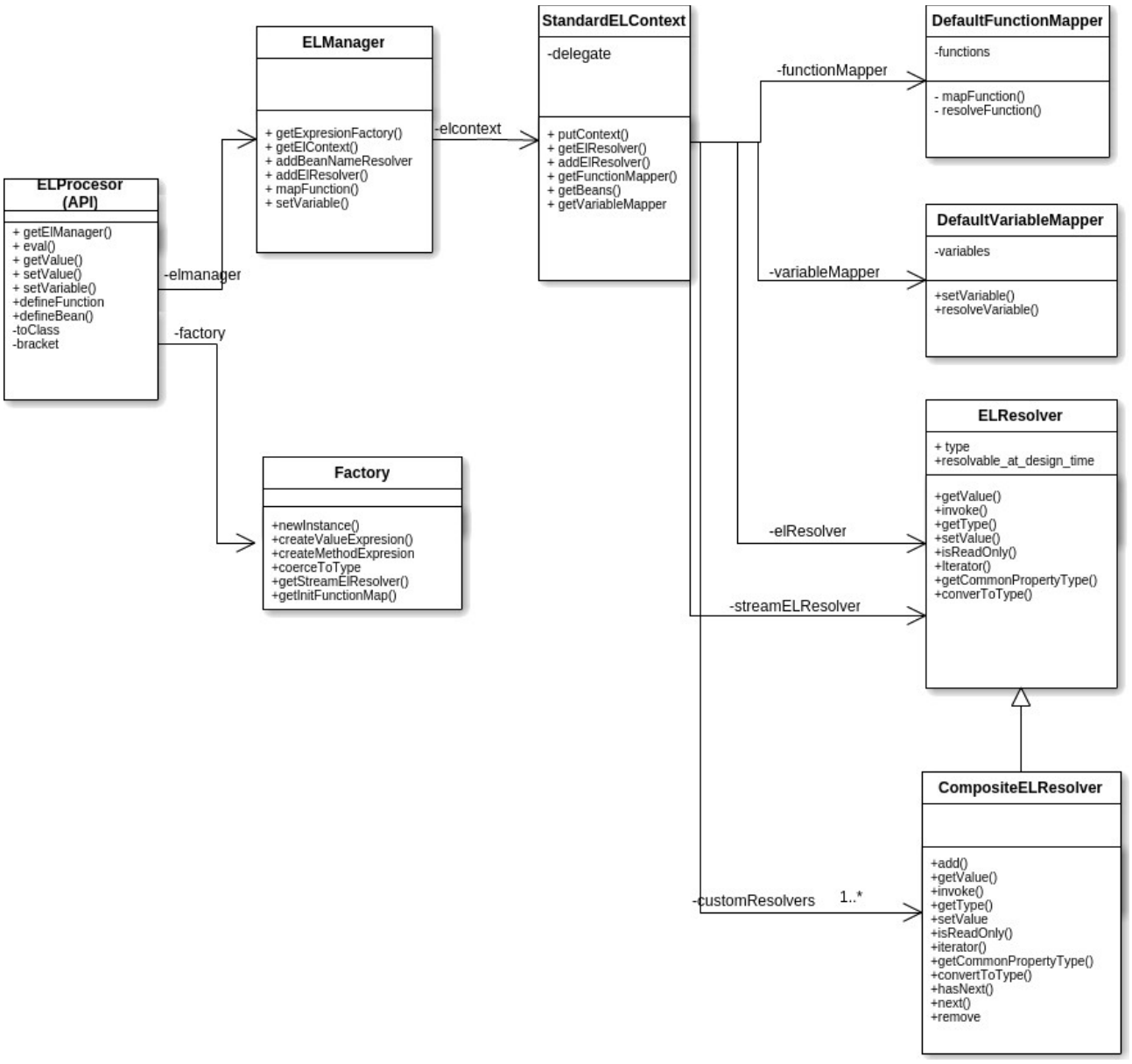

Figura 6.1: El diagrama de clases que se debe especializar para utilizar el lenguaje EL de la especificación JSR-341 v3.0.

En este contexto, la implementación del módulo de expresiones consistió en el desarrollo de un conjunto de clases que permite utilizar una porción de toda esta potencia y expresividad, definida en la especificación, donde la porción utilizada era aquella que tenía sentido para el contexto en el que el módulo se iba a instanciar y la necesidad para la que fue desarrollado.

Desde un principio, se planteó la idea de la reusabilidad como una característica muy necesaria, ya que los casos de uso para el lenguaje así lo ameritan. Por esto, se pensó en el desarrollo como un módulo dentro DSpace, como parte de las extensiones del sistema. De esta forma, se lo puede instanciar desde casi cualquier punto en la ejecución de DSpace, ya sea dentro de la ejecución de una tarea de curación (ver *capítulos 3 y 7*) o en el momento de devolver los datos mediante el *data provider [(DuraSpace, 2012c)]*. De ahí, surge el nombre de módulo de expresiones.

Por lo dicho hasta aquí, el módulo debería encapsular todo el funcionamiento correspondiente a la evaluación de las expresiones, la toma de decisiones sobre dichas expresiones y debería dar métodos para analizar los resultados obtenidos. También, debería permitir algún tipo de instanciación, para que los objetos que se puedan navegar sean instancias existentes durante la ejecución y no solamente clases estáticas de un metamodelo predefinido. Con el análisis de las herramientas escogidas realizado, se pensó en un diseño para la solución.

Diseño

El módulo de expresiones se pensó como una API que actuaba de punto de entrada único para los usuarios del lenguaje. Encapsula el comportamiento del lenguaje de expresiones definido en la JSR-341 y lo simplifica, además de instanciar el metamodelo, con los objetos necesarios para realizar la evaluación de la expresión. En el módulo de expresiones se realizan los chequeos de permisos necesarios, se registran en archivos de log los resultados y se instancian los objetos del lenguaje para que se pueda utilizar. Por tener un estado interno, no se trata de un objeto *Fachada*. No obstante, la necesidad de utilizar el módulo de manera concurrente, podría surgir en un futuro, por lo que habría que adaptar la clase para que mantenga la consistencia frente a posibles utilizaciones concurrentes. (Ver sección *Trabajos Futuros*, capítulo 8).

Se diseñó un módulo que guarda una referencia a una instancia de la clase *ELProcesor* y al objeto encargado de generar una instancia del metamodelo. A partir de la creación de un objeto *ExpresionModule*, se pueden definir objetos como parte del modelo, asociar funciones y establecer valores para variables, todo esto utilizado dentro de una expresión en el lenguaje, que se puede ejecutar a través del método *eval()*. Además, el método *eval* devuelve un Objeto resultado, a lo que es posible aplicarle un *down-casting [(Oracle, s. f.-b)]* de acuerdo a la situación (a boolean si es una validación o transformación, a algún objeto DSpace si se trata de una selección). También se estableció un tipo especial de excepción, que especializa a las excepciones nativas de java de la clase *RuntimeException [(Oracle, s. f.-c)]*. De esta forma, el módulo encapsula y reporta los errores que pueden surgir durante la instanciación. El usuario recibe los errores de parseo, las excepciones de DSpace y las posibles dificultades durante la ejecución de las expresiones, de manera amigable y persistida en el log del sistema. En la *figura 6.2* se puede ver un diagrama de clases que ayuda a comprender el diseño del módulo.

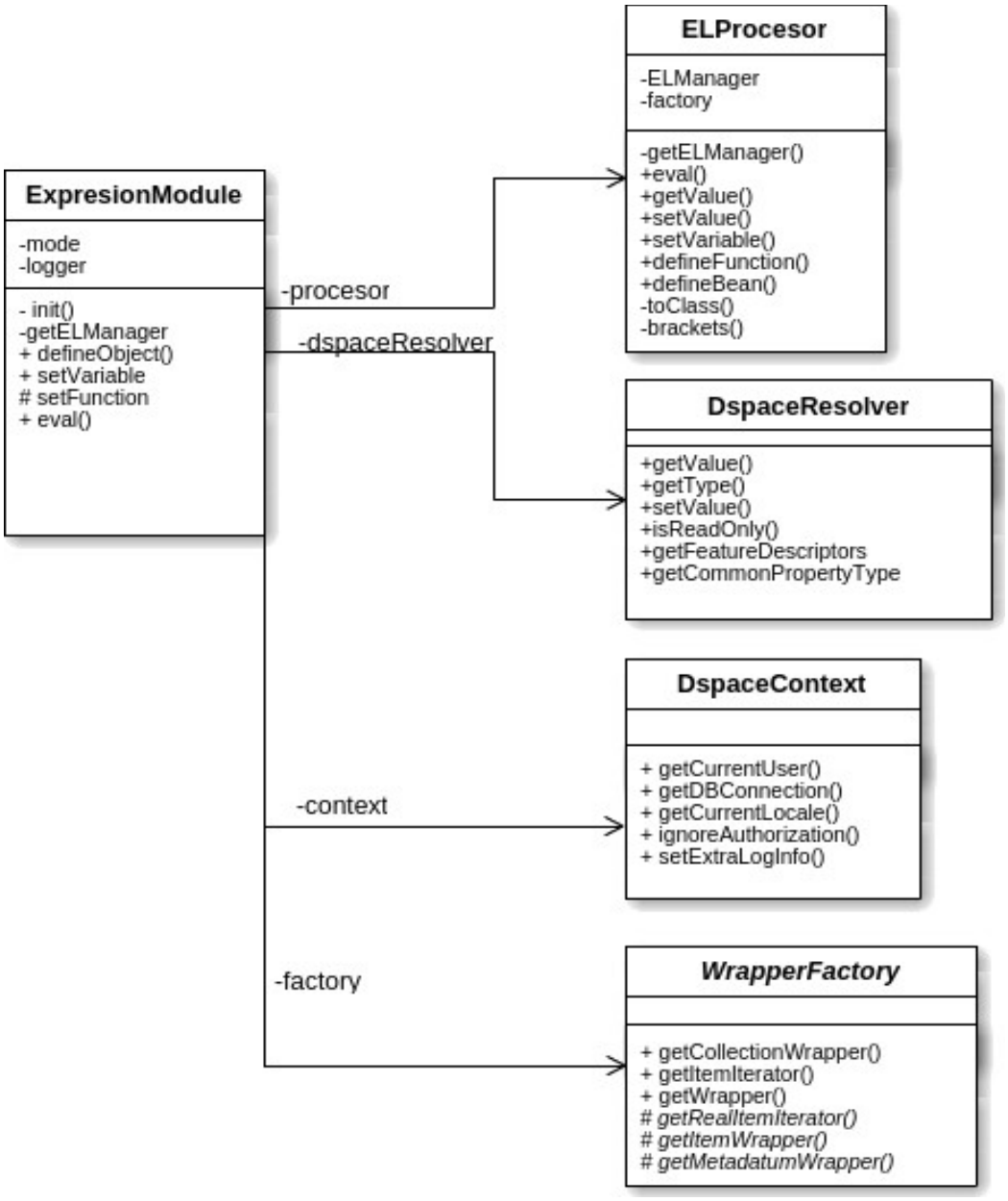

Figura 6.2: modelo UML simplificado del módulo de expresiones diseñado para implementar la herramienta de validación.

Niveles de abstracción

Una vez realizado el estudio de la JSR-341, se comprendió la expresividad que poseía el lenguaje construido con la herramienta, de manera nativa. Para aprovechar esta expresividad en el contexto de los repositorios digitales, se instanció el lenguaje de expresiones con el modelo semántico definido con anterioridad para el lenguaje (Ver capítulo 5, sección “Modelo Semántico”).

A pesar de la potencia de la herramienta utilizada, que incluso permitía acceder a los métodos mutadores de los objetos correspondientes al modelo de DSpace (*ver en capítulo 3*), requirió plantear formas seguras para instanciar el metamodelo, a la hora de evaluar expresiones construidas con el lenguaje.

Para tener control sobre las operaciones realizables en distintos objetos del metamodelo se planteó la utilización de objetos con cierto nivel de abstracción, representado por una jerarquía, sobre el modelo de DSpace, que controlan el acceso a las operaciones sobre los objetos del sistema, de acuerdo al nivel de seguridad que se pretende otorgar. De esta forma, se plantearon tres capas de abstracción probables para representar tres niveles de acceso distintos:

El primer nivel sería el más básico, pero a su vez el más necesario, el nivel para "*operaciones de sólo lectura*". Este nivel debería realizar tanto las operaciones de validación como de selección en el lenguaje e implementar aquellos casos de uso planteados para la herramienta, que no involucran transformaciones.

En el segundo nivel, deberían incorporarse las transformaciones sobre objetos del sistema. Se añade una capa para controlar la seguridad (la habilitación de perfiles de usuarios de acuerdo a los permisos que posea) y la consistencia (que una transformación deje al sistema en un estado consistente, por ejemplo mediante el uso de transacciones).

Por último, el tercer nivel representaría el "*acceso completo*" a los objetos del sistema DSpace, permitiendo todas las operaciones que el sistema puede realizar sobre dichos objetos, pero desde el lenguaje. En la *figura 6.3,* se puede apreciar el diseño planteado.

Figura 6.3: los distintos niveles de autorización planteados para el módulo de expresiones.

Para establecer estos tres niveles se utilizó el patrón de diseño "*AbstractFactory*"[(Gamma et al., 1995)], y se planteó una clase que instancia objetos que envuelven a los objetos DSpace con los que se quiere trabajar con el lenguaje. De esta forma, tanto el *Factory* como los objetos envolventes o *"Wrappers"* son los que definen el nivel de acceso.

La idea principal es tener una clase “Factory” que subclasifique a la clase *AbstractFactory* y que implemente los métodos *getWrappers()* necesarios para obtener los objetos envolventes de acuerdo al nivel de acceso. De la misma forma, para respetar la jerarquía de capas planteada, los wrappers de un nivel de acceso mayor, deben subclasificarse de los wrappers de un nivel menor, para asegurar que la interfaz de cada nivel se mantenga en el siguiente. El UML en la *figura 6.4* representa el modelo de clases correspondiente, con el nivel de “sólo lectura” implementado en su totalidad. Cabe destacar que también se da soporte para colecciones de objetos DSpace y para los iteradores de ítems, por lo que el modelo de wrappers los tiene en cuenta.

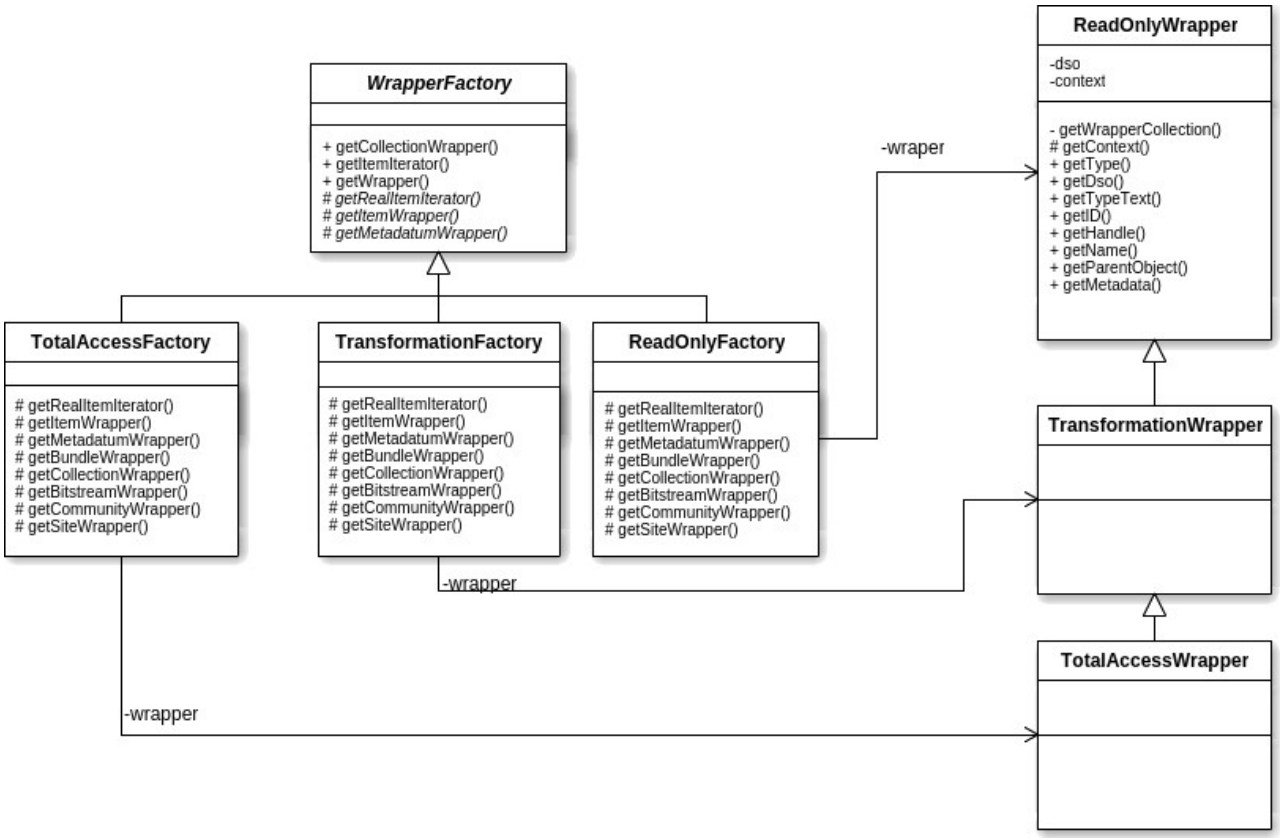

Figura 6.4: el UML en una versión simplificada para representar los distintos tipos de factory y su correspondencia con los Wrappers.

Con esto, de acuerdo al nivel en donde se instancie el módulo de expresiones, será el *Factory* que utilizará y, por consiguiente, los wrappers que serán instanciados durante el proceso. Por consiguiente, el módulo tendrá un control exhaustivo sobre la seguridad de las expresiones evaluadas y de los resultados obtenidos, independientemente del lugar en donde se lo instancie y el usuario que lo utilice.

Con un diseño preestablecido y fundamentado, se procedió a definir el foco de la implementación, delimitado por las necesidades que se debían atender con más prioridad y las clases base ya mencionadas.

Con la idea de plantear el desarrollo de la herramienta con un enfoque iterativo e incremental, se optó por una metodología orientada a casos de uso, donde se pensaron posibles casos de uso básicos para el lenguaje y se desarrolló lo necesario para cubrirlos. Para marcar una línea de corte en el alcance de este trabajo, se decidió desarrollar hasta el primer nivel de acceso al módulo de expresiones, permitiendo métodos de *“sólo lectura”* y cumpliendo

con el objetivo de implementar las operaciones de selección y validación del lenguaje para repositorios digitales.

Desarrollo del Módulo de Expresiones para DSpace

En el módulo Additions de DSpace se desarrollaron las clases correspondientes al módulo de expresión, el *DSpaceResolver,* la clase abstracta *WrapperFactory* y la subclasficación para *“read only”*.

A partir de aquí, se escribieron, a medida que eran requeridas, las clases *“read only”* como *wrappers* de cada uno de los objetos del metamodelo. De esta forma, se aseguró que el desarrollo estuviese acotado y enfocado. A continuación, se nombran una serie de expresiones en el lenguaje planteado, que sirvieron para localizar distintas problemáticas a resolver durante la fase de desarrollo del módulo. Las mismas, están clasificadas de acuerdo a la operación genérica de la que se desprenden, para facilitar el entendimiento de cada una.

Expresiones de validación

La primer operación de validación planteada fue ”*Verificar que un Bundle sea el <<Original>>*” ; la expresión en el lenguaje que representa esta operación plateada y que el módulo pudo resolver, es:

bundle.name == ‘ORIGINAL’

La resolución de la operación consistió en la creación de “*ReadOnly” Wrappers* para los objetos del metamodelo. Tal es así que, a partir de este ejemplo se implementaron *Wrappers* para los Ítems, las Colecciones, las Comunidades, los Bitstreams, los Bundles y los Metadatos en DSpace. (Ver capítulo 3, sección “*Modelo de DSpace*”) . Además, se añadió un método extra a cada Wrapper, que permite devolver de forma más amigable los metadatos (encapsulados en un *MetadatumWrapper*), para poder realizar validaciones sobre estos. Como consecuencia, se pueden utilizar expresiones del estilo *dso.getMetadata(“Un_metadato”)* en el lenguaje.

Para validar que el módulo de expresiones pudiera resolver la operación planteada luego del desarrollo de los wrappers adecuados, se crearon validadores ocasionales, utilizando la librería JUnit [(Massol & Husted, 2003)]. Se estableció un paquete para pruebas que realizaba distintos chequeos.

A partir de aquí, el módulo permitió resolver muchas otras validaciones en el lenguaje, como por ejemplo:

- “Validar que un item tenga un identificador persistente” :
  - !empty(item.handle)
- “Verificar que un artículo posea autor”:
  - item.getMetadata.stream().anyMatch(m->m.name == “dc.creator”)
- “Verificar que un DSpaceObject sea un bitstream”:
  - dso.typeText == ‘BITSTREAM’
- “Verificar que un bitstream tenga un algoritmo de chequeo asignado”
  - empty(bitstream.getChecksumAlgorithm())
- “Validar que una colección pertenezca a una comunidad dada en *’community’* ”
  - collection.communities.stream().anyMatch(c-> c.ID == community.ID)

Expresiones de selección

La segunda operación posible para el lenguaje que cabe mencionar en esta sección es “*Mostrar todos los Items de una Colección, que tengan una fecha de publicación (dc.date.issued)*”. La expresión en el lenguaje que representa esta operación es:

collection.items.stream().filter(i->empty(i.getMetadata("dc.date.issued"))

Para poder resolver esta operación, que se trataba de un caso específico de selección sobre una colección determinada de *ítems*, se debió implementar un Wrapper para las colecciones de objetos DSpace, el *CollectionWrapper*. Esta clase resultó muy conveniente para permitir la interacción entre distintos wrappers de diferentes objetos de DSpace. De esta forma, cada vez que a un objeto contenedor de otro (colecciones de comunidades, ítems de bundles, etc) se le piden los objetos contenidos, el wrapper que encapsula el comportamiento de dicho objeto, en realidad devolverá o un wrapper del objeto contenido o un *CollectionWrapper* de un conjunto de estos objetos. Dicho de otro modo: si B es una colección y A es un objeto y al wrapper A' de A se le solicita B, se obtendrá o bien un wrapper B' de B o bien un CollectionWrapper de B'_1, B'_2, B'_3...B'_n, donde B'_i encapsula al objeto B_i original. Una vez desarrollada la clase, se volvió a chequear su funcionamiento con nuevos testeos implementados con JUnit.

El tercer caso de uso que es necesario citar es "*Mostrar todos los DSpaceObjects del sistema, que cumplan con un tipo determinado (ej:todas las colecciones)*". Lo que en el lenguaje se podría escribir como:

site.dso.stream().filter(d->d.type == 'COLLECTION')

Para dar soporte a este caso, se necesitaron objetos más generales. Es decir, algunos objetos que contengan a los demás, sobre los cuales realizar la selección. Como consecuencia de esto, se crearon el *ContextWrapper* y el *SiteWrapper*, para encapsular los objetos *Context* y *Site* de DSpace respectivamente. Esto fue funcional para resolver el caso antes mencionado, donde se tiene ya definido el tipo de los elementos que se quieren listar. No obstante, se ha comenzado a plantear la necesidad de algún mecanismo que permita resolver operaciones que requieran referenciar elementos genéricos del modelo, por ejemplo, por medio de un valor literal, que podría ser el *Handle*. Esto ha quedado como una posible extensión y continúa evaluándose en la actualidad *(Ver sección "Trabajos Futuros", Capítulo 8)*.

Con este modelo de clases, el Módulo ya es capaz de resolver todos los casos de uso que involucran validaciones, proyecciones (selecciones de un solo elemento como resultado) o la mayoría de las operaciones de selección más complejas. Algunos ejemplos de las operaciones válidas para la herramienta pueden ser:

- "Listar todos los metadatos en español de un item":
  - item.getMetadata().stream().filter(m->m.language == 'ES')
- "Listar todos los bundles vacíos de un item":
  - item.bundles.stream().filter(b->empty(b.bitstreams))
- "Listas los bitstreams de un item que no tengan MD5 como función de checksum"
  - item.bundles.stream().forEach(b->b.bitstreams.stream().filter(bits->bits.getChecksumAlgorithm()!='MD5'))

No obstante, se encontraron algunos casos donde la eficiencia del módulo se veía dañada por la gran cantidad de objetos en memoria que se necesitaban para resolver la evaluación de la expresión. Por ejemplo, el caso de uso : "*Listar todos los items de una comunidad determinada*" (*community.collections.stream().foreach(c->c.item)*), generaba una gran carga en memoria al instanciar cada *ItemWrapper* de cada Colección en una Comunidad.

Además, por un caso puntual de implementación, el mensaje *'getItems()'* de la clase Colección del modelo de DSpace, que se utiliza para resolver la fracción de la expressión

correspondiente a *‘c.item’*, devuelve un objeto especial *itemIterator* [(Doxigen, s. f.)] que no implementa la interfaz *Collection*, ni la interfaz *Iterator*, nativas de Java [(«java.util (Java Platform SE 7)», s. f.)], lo que imposibilita la utilización de colecciones de Items con el lenguaje. Esto se daba debido a que los operadores de colección que venían por defecto en el lenguaje de expresiones (stream(), map(), filter(), count(), entre otras, ver más arriba) esperan que el objeto al cual aplicar el operador sea o un arreglo nativo o una subclase de *Java Collecction,* sobre la cual se pueda iterar.

No obstante, el uso de esta clase iteradora en DSpace, era razonable ya que mediante la misma se podían navegar todos los items de una colección sin la necesidad de que estén todos instanciados en memoria al mismo tiempo, o dicho de otro modo mediante un proxy de almacenamiento [(Gamma et al., 1995)] implementado por DSpace.

Por todo esto, se implementó un nuevo *Wrapper*, esta vez para la clase ItemIterator, que sí subclasifica la clase abstracta nativa de JAVA ‘Collection’ y que con el mensaje *getIterator()* devuelve un ItemIteratorWrapper, que encapsula un ItemIterator de DSpace y, por lo tanto, mantiene la eficiencia del manejo de los elementos en memoria.

Gracias a esto, los casos de uso ineficientes mejoraron considerablemente, y las operaciones sobre colecciones nativas del lenguaje de expresiones, ahora también se pueden aplicar sobre las colecciones de items DSpace. Todo esto validado con los test de JUnit correspondientes. Como ejemplo de lo mencionado en el párrafo anterior, ahora se pueden realizar operaciones complejas sobre colecciones de Ítems en el lenguaje, del estilo de las siguientes:

- “Listar todas las colecciones vacías de una comunidad”:
  - `community.collections.stream().filter(c->!empty(c.items))`
- “Proyectar todos los libros que no poseen ISBN en una colección”:
  - `collection.items.stream().filter(i-> i.getMetadata(“dc.type”) == ‘BOOK’ && empty(i.getMetadata(“sedici.identifier.isbn”)))`
- “Listar todos los bundles ‘Original’ de una colección”:
  - `collection.items.stream().forEach(i->i.bundles.stream().filter(b->b.name == ‘Original’))`

Configuración y Ejecución

Para poder utilizar el módulo de expresiones se debe estar en el contexto de una ejecución de DSpace. Luego seguir los siguientes pasos:

- Crea una instancia de la clase *ExpresionModule* y enviarle como parámetro un objeto de clase *Context* de DSpace (ver *Capítulo 3*) que representa el contexto general donde se instancian los distintos objetos del metamodelo.

Por el momento, el módulo sólo admite el modo *“solo lectura”*, donde se pueden realizar validaciones y selecciones con las expresiones del lenguaje. Sin embargo, en el futuro se piensa implementar el soporte para las operaciones de escritura y, por consiguiente, permitir a las transformaciones (ver capítulo 8, sección *Trabajos Futuros*)

- Con la instancia del módulo de expresiones ya definida, configurar variables o funciones estáticas para el lenguaje utilizando los métodos *defineFunction()* y *defineVariable()*, que reciben strings con la referencia y la expresión por la cual reemplazarla, que se evaluará primero, y cuyo resultado representará el valor real de la variable/función correspondiente, durante la evaluación de la expresión que la referencia. [(Juneau, 2013)]

- Finalmente, mediante el método *defineObject()*, instanciar las distintas referencias hacia objetos del contexto DSpace actual en donde se está ejecutando el módulo. Por ejemplo: un generador de reportes que debe hacer selecciones sobre una comunidad, coleccion o item determinado, incluso sobre bundles, puede ejecutar *defineObject("Referencia",ObjetodeContexto)* y luego en su expresión del lenguaje referirse a las propiedades y mensajes públicos del *ObjetodeContexto*, a través de la "Referencia". De forma más clara, si se quiere preguntar por todos los Items de la Colección A, se puede hacer defineObject("A",Coleccion A) y, desde el lenguaje *A.getItems()*, haciendo referencia al método getItems que viene definido en la clase Colección(*definida en capítulo 3*) de DSpace.

En el capítulo siguiente se muestra un caso de ejemplo de la utilización del Módulo de expresiones, que responde a la mejora de los controles automáticos de calidad en el repositorio institucional de la UNLP. Como un caso concreto y puntual de lo establecido hasta el momento en este trabajo, se muestran algunas *tareas de curación definidas (ver sección "Tareas de curación" en capítulo 3)*, que utilizan el módulo y los resultados obtenidos al implementarlas y ejecutarlas.

Capítulo 7 | Ejemplo de uso del Módulo de Expresiones: Tareas de Curación

Introducción

Una vez completa la fase de desarrollo del Módulo de Expresiones para Dspace, resultó necesario definir un contexto de utilización. Tras un análisis de costo-beneficios entre los distintos usos posibles planteados, se decidió utilizar el módulo a través de la definición de nuevas tareas de curación para el sistema. Del mismo modo, la decisión precedente también estuvo apoyada en trabajos previos vinculados a tareas de curación dedicadas a la preservación de los ítems del repositorio. [(Terruzzi et al., 2014)]

En este contexto se plantearon una serie de tareas de curación que utilizan directamente el módulo de expresiones, instanciándolo y validando distintas expresiones que conllevan comprobaciones y selecciones sobre el modelo de datos de DSpace (Ver capítulo 3). Las nuevas tareas de curación fueron añadidas en un nuevo paquete en el *módulo Additions* (Ver capítulo 3, sección “Módulo Additions”), para que se puedan crear de manera incremental (implementar a medida que sea necesario) y para que puedan ser instanciadas desde distintos contextos del sitio.

Definición del Modelo de Tareas Planteado

Se planteó un modelo de tareas con clases concretas que subclasifican la clase abstracta AbstractCurationTask (Ver capítulo 3, sección *“Tareas de curación”* para justificación). Cada tarea planteada dentro del paquete *ModuleExpresionTaks* representa una operación general (*capítulo 5, “requisitos funcionales”)* posible planteada para el lenguaje e implementada con el módulo de expresiones. De esta forma, se pensó en al menos tres tareas de curación factibles : *ValidationTask*, *SelectionTask* y *TransformationTask*. Lógicamente, debía también representarse la posibilidad de realizar distintos tipos de validaciones, selecciones o transformaciones, esto se implementó utilizando el patrón de diseño *Template Method* [(Gamma et al., 1995)], de forma tal que cada una de las clases mencionadas define un método plantilla que expone una serie de pasos a realizar para hacer una validación, una selección o una transformación. Luego, estos pasos son especializados de acuerdo a los requerimientos del modelo para evaluar la expresión. Esto es llevado a cabo por clases java relativamente sencillas que son subclases de alguna de las tres planteadas con anterioridad, y que sólo deben definir sobre cuál o cuáles DspaceObjects se aplican, la expresión en el lenguaje a evaluar y cómo expresan su resultado. Por ejemplo: la tarea de validación que comprueba que las colecciones no estén vacías (NonEmptyCollectionTask), define su aplicabilidad sólo sobre Collections de Dspace; define la expresión del lenguaje que va a validar con cada colección, para realizar este chequeo y define que, lógicamente por tratarse de una validación, el resultado será un valor Booleano (*Falso/Verdadero*), de acuerdo a si tuvo o no éxito. Para esclarecer esto se muestra la *figura 7.1* con el modelo de tareas de curación planteado.

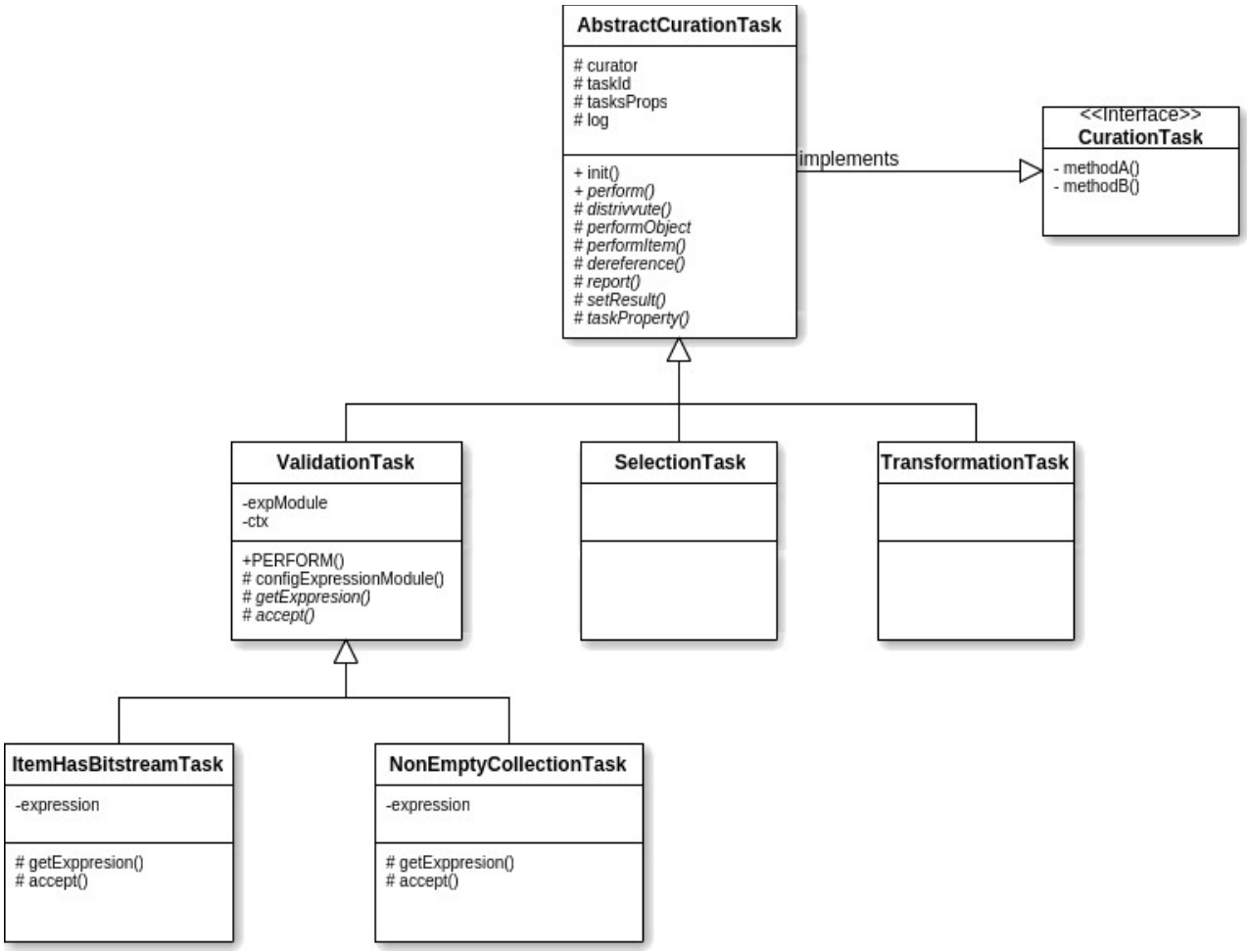

Figura 7.1: un UML aproximado de la jerarquía de tareas de curación implementada, para facilitar su utilización y el costo de su desarrollo, a partir de la reutilización de componentes en común.

Cabe destacar que, si bien la utilización de este modelo tiende a simplificar la implementación de posibles tareas de curación que usan el lenguaje a partir de expresiones, se plantea un esquema 1:1 con las expresiones propiamente dichas, por lo que pensarlo en un entorno incremental, requeriría el desarrollo constante de tareas de curación que evalúen expresiones distintas. Para resolver esto, se propone como trabajo futuro, la posibilidad de implementar tareas que evalúen expresiones definidas en tiempo de ejecución (*Ver Capítulo 8, trabajos futuros*).

Ejemplos de Tareas Implementadas

Para ejemplificar el modelo descrito en la sección anterior, se definen dos tareas de curación completamente implementadas y que, en la actualidad, son ejecutadas por los administradores del Repositorio Institucional de la UNLP.

Como ya se dijo, el módulo de expresiones posee en la actualidad, la capacidad para responder a consultas que no requieran permisos especiales y autenticación, ni que se deban realizar de manera atómica *(Ver Capítulo 8, sección “Trabajos futuros”)*.

Por esto, las tareas implementadas extienden la clase *ValidationTask,* ya que evalúan expresiones que realizan validaciones sobre el modelo de Dspace.

De esta forma, la tareas deben implementar los métodos específicos que define dicha clase. Estos métodos son:

- *accept(dso DspaceObject):* debe devolver un valor Booleano (Verdadero/Falso) que indica si el DspaceObject es aceptado por la tarea, en general porque posee la clase del modelo DSpace (Ver capítulo 3) aplicable a la tarea en cuestión.
- *getExpression():* devuelve la expresión a utilizar, la misma DEBE ser una validación, por lo que se espera un Booleano como resultado de su evaluación.

Tarea de colección no vacía (NonEmptyCollectionTask)

Tal y como su nombre lo indica, esta tarea se diseñó para analizar una colección dada y verificar si posee al menos un item. El nivel de ejecución de esta tarea puede ser el de “Sitio completo”, debido a que se pueden localizar colecciones defectuosas o que han quedado obsoletas en el repositorio.

El modelo planteado permite que la programación de esta tarea haya quedado en la formulación de la expresión adecuada en el lenguaje de expresiones implementado. Esto se dió así por dos motivos: en primer lugar, la lógica necesaria para solucionar el problema quedó derivada en el Módulo de Expresiones. En segundo lugar, toda la complejidad que añade la utilización de este módulo, fue encapsulada por la tarea padre que realiza la validación y plantea el *Template Method*. De este modo, la expresión que realiza la validación buscada es la siguiente:

```
!empty(dso.getItems())
```

Donde *dso* es una variable que contiene una DSpaceObject del tipo Collection

Tarea de Bitstreams por Item (ItemHasBitstreamTask)

Esta tarea se implementó para responder a la necesidad de un mecanismo para localizar ítems en el repositorio, que no tengan ningún archivo asociado en ninguno de sus Bundles, debido a esto su nombre real puede ser autodocumental. Este tipo de validaciones era costosa de llevar a cabo, incluso con la potencia del entorno de curación planteado por DSpace, debido a la necesidad de comprobar en cada uno de los bundles, la existencia de un bitstream asociado; tareas que insumen una gran costo (tanto en duración de la implementación como en memoria y tiempo de ejecución); asociado a que el sistema de reportes propuestos por DSpace para curación, está pensado a nivel Ítem, lo que hacía muy difícil obtener resultados amigables para el usuario o sistema que realizaba los chequeos. En este contexto, el Módulo de Expresiones permitió disminuir el costo de su implementación y obtener una mejora en eficiencia y tiempo de ejecución. Asociado a lo precedente, la implementación permite obtener reportes a nivel Ítem más exactos, lo que debería facilitar la interpretación de la información obtenida al ejecutar la tarea.

Todo lo anterior es posible mediante la formulación de una expresión adecuada, ejecutable con el módulo de expresiones y que haga la comprobación planteada para la tarea. En este caso, la expresión construida fue:

```
dso.bundles.stream().anyMatch(b->!empty(b.bitstreams))
```

Donde *dso* es algún DSpaceObject del tipo Item asignado a la tarea de curación.

Nótese que las tareas de curación aquí descritas requieren operaciones sobre el módulo en modo “solo lectura”, esto se realizó intencionalmente, para obtener una evaluación de la eficiencia y eficacia del modelo implementado (que se muestran en la siguiente sección) e

intentando obtener retroalimentación, para definir las mejoras a plantear en el corto y el mediano plazo. (Más información en trabajos futuros, capítulo 8)

Ejecuciones de ejemplo

Para llevar a cabo las tareas de curación se utilizó el marco de trabajo propuesto por DSpace y se las ejecutó desde un intérprete de comandos, empleando la interfaz predefinida por el software para este tipo de práctica [(Terruzzi et al., 2014)]. Por esto, las tareas antes mencionadas se implementaron con un sistema de reportes que encapsula los errores que pueden aparecer (tanto los del sistema, como los correspondientes al módulo de expresiones) y los reporta de la forma predefinida por las clases de curación de DSpace. [(DuraSpace, s. f.-d)]

La ejecución se realizó en dos entornos distintos. En primer lugar se probaron las tareas en un sistema que representaba una versión reducida del entorno real, para probar la eficacia de las tareas y corregir posibles errores indeseados. Cuando el funcionamiento adecuado, las tareas se llevaron al entorno real y se las ejecutó primero para un elemento, luego para una colección de elementos y finalmente en todo el repositorio. Se obtuvieron varios resultados prometedores, que se muestran a continuación.

NonEmptyCollectionTask

Se ejecutó la tarea *noEmptyCollection* para una sola colección. La validación se realizó correctamente y los costos en memoria y tiempo fueron mínimos. Se probó entonces la misma tarea en una comunidad completa, esta ejecución también fue exitosa. Se pasó entonces, a probar la tarea en todo el repositorio, evaluando los costos en memoria y en tiempo, además de los resultados obtenidos y los reportes generados. La validación pudo llevarse a cabo sin problemas. La *figura 7.2* muestra la consola y un ejemplo de ejecución de la tarea.

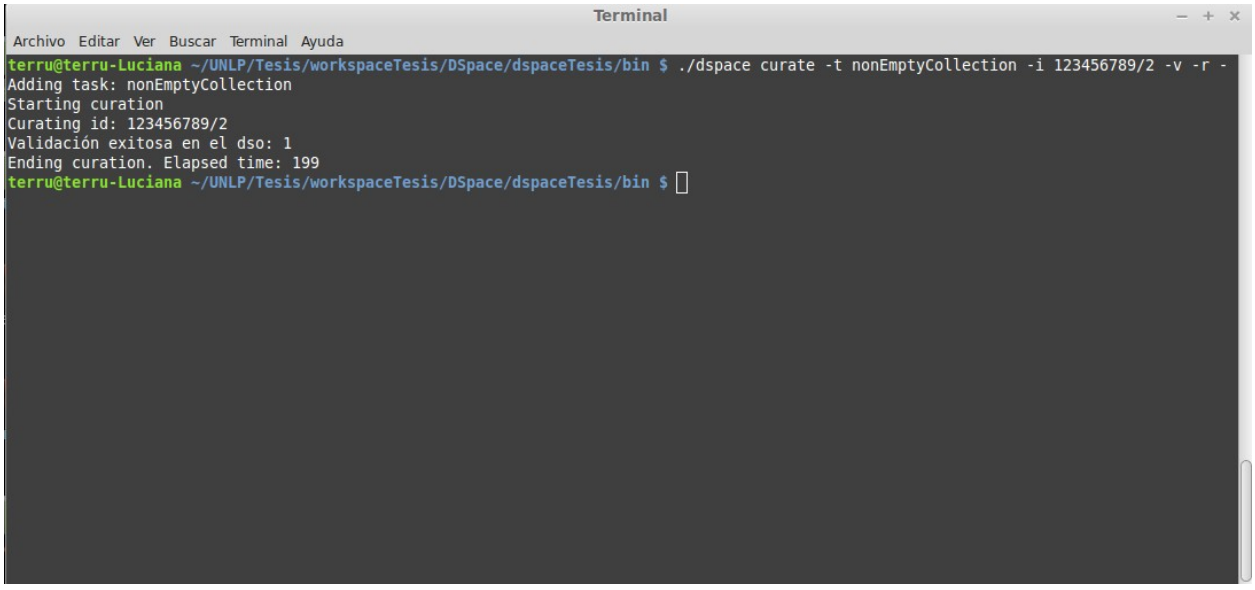

Figura 7.2: una ejecución de la tarea nonEmptyCollection mediante la interfaz de consola de DSpace

itemHasBitstreamTask

Se ejecutó la tarea *itemHasBitstreamTask* para un solo item. La validación se realizó con éxito y, de nuevo, los costos en memoria y tiempo fueron mínimos. Se procedió entonces a

realizar la operación con una colección completa (que es el alcance requerido en el repositorio), los costos fueron muy buenos y las validaciones correctas, es decir que se realizó la tarea necesaria y con el nivel de alcance apropiado.

La *figura 7.3* muestra el resultado de una ejecución de ejemplo, donde se puede apreciar que el tiempo ha sido de un umbral razonable y los resultados de las validaciones se mostrado con éxito.

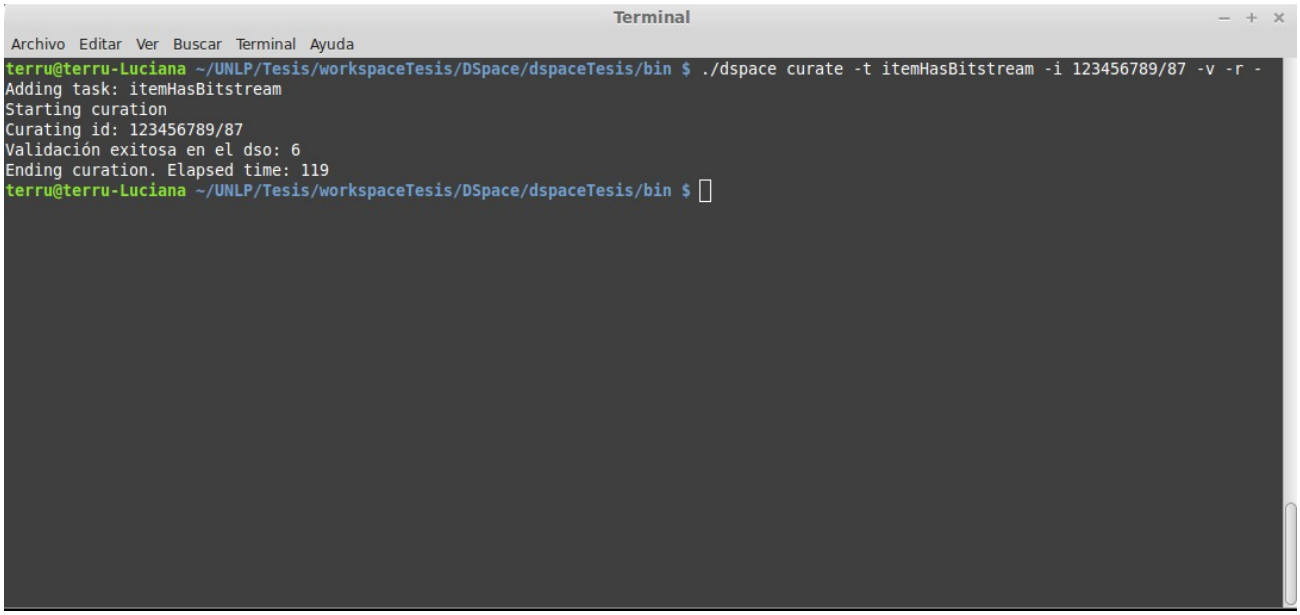

Figura 7.3: una ejecución exitosa de ejemplo de la tarea itemHasBitstream implementada

Las validaciones realizadas han encontrado errores en distintos elementos, que se continúan corrigiendo en la actualidad. Además, más adelante se analizarán las consecuencias positivas que han resultado del modelo de curación planteado aquí, que se puede destacar por su simplicidad en el diseño, y la implementación minimalista de tareas, que se pueden construir con relativa facilidad, aprovechando toda la potencialidad del lenguaje. (Ver conclusiones, capítulo 8)

Capítulo 8 | Conclusiones y Trabajos futuros

Conclusiones

El desarrollo definido en este trabajo puede verse como una implementación en capas, por esta razón, se considera apropiado separar las conclusiones consistentes con los distintos niveles de abstracción planteados.

Especificación de la Herramienta

Con este trabajo se buscó realizar un aporte específico pero práctico, que respondía a una necesidad vigente en el repositorio institucional de la UNLP. Una de las contribuciones de esta tesina consiste en la definición de un lenguaje que posee una sintaxis abstracta y concreta lo suficientemente ortogonal como para dar lugar, con algunos cambios y cotas, a posibles implementaciones en otros repositorios, sobre otros contextos y/o con distintas herramientas. La implementación de gran parte de la herramienta fue realizada aplicando patrones propios del ámbito de lenguajes específicos de dominio, que buscan normalizar las fases decisión, diseño e implementación de este tipo de desarrollos. La metodología utilizada garantiza un cierto nivel de abstracción sobre la implementación realizada en SEDICI, lo que permite otras implementaciones para el mismo lenguaje aprovechando las características de su especificación, que se adapten a otros casos de uso y resuelvan otros objetivos en el dominio de los repositorios. En la sección "*Trabajos Futuros*" se muestra un posible ejemplo de esto.

En lo relativo al objetivo inicial planteado en cuanto a expresividad, se ha logrado describir un lenguaje lo suficientemente expresivo como para integrar y representar todas las entidades presentes en un repositorio digital, gestionado por el sistema DSpace; esto se puede observar a través de la sintaxis abstracta propuesta y los elementos presentes en el metamodelo. No obstante, la definición de un lenguaje lo suficientemente genérico para representar los contenidos de cualquier repositorio digital, partiendo quizá de un modelo abstracto (se menciona en : [(Texier, De Giusti, & Gordillo, 2014)]) de repositorio, es un problema complejo que requiere de un análisis más profundo y que escapa a los lineamientos de esta tesina.

Implementación del lenguaje: Módulo de expresiones

La implementación del módulo de expresiones requirió un análisis sobre las formas de instanciar la herramienta utilizada y de adaptarla para aceptar el modelo semántico propio. Aunque gracias a esta implementación se logró cumplir el objetivo de desarrollar una herramienta de control dinámica que permite realizar validaciones y generar reportes, sin necesidad de conocer demasiado sobre las implementaciones del software Dspace, la expresividad de la solución desarrollada en este trabajo quedó definida por la sintaxis concreta de la especificación JSR-341, sumada a la del modelo semántico. Este puede verse como un punto de mejora, ya que se podría conseguir un lenguaje más cercano a la sintaxis abstracta planteada en la especificación de la herramienta (capítulo 5), lo que lo haría más sencillo y genérico, si se consideraran otras formas de implementación, por ejemplo, planteando la implementación como DSL externo. Este punto se discutirá en la sección "Trabajos Futuros".

La decisión de definir un módulo independiente del núcleo de DSpace por medio del módulo *additions* ha resultado positiva en el desarrollo. Se obtuvo un módulo de expresiones encapsulado que puede ser referenciado desde cualquier punto de ejecución del código Dspace y que es capaz de utilizar todos los componentes del sistema subyacente. Sin

embargo, el grado de reutilización y de ampliación del módulo podrá evaluarse en los trabajos futuros, puesto que se estudiará la facilidad con la que se puede expandir la implementación.

Caso de Uso: Tareas de curación

El objetivo principal planteado para este trabajo fue ”permitir la utilización de *mecanismos de control de calidad dinámicos en un repositorio institucional.*”, por lo que un caso de uso como las tareas de curación resultó útil para dimensionar la eficacia de la solución y la eficiencia de la implementación, acorde con el objetivo planteado.

Desde un punto de vista técnico, cabe mencionar que gracias al encapsulamiento que plantea el módulo, las tareas de curación que lo utilizan resultan más simples de implementar, es decir, el costo en tiempo y aprendizaje que ameritaban (como se puede apreciar en [(De Giusti et al., 2013) y (Terruzzi et al., 2014)]) ha disminuido considerablemente, siempre que se tengan conocimientos en la sintaxis de la herramienta diseñada.

El modelo simple y reutilizable planteado para las tareas de curación que emplean el módulo de expresiones, permite la implementación de estas tareas de una forma relativamente sencilla, haciendo que la complejidad de las validaciones que dichas tareas deben realizar quede encapsulada al hecho de construir la expresión correcta en el lenguaje planteado.

Por último, es propio enunciar que a partir de la ejecución de las tareas de curación se observó que las mismas hacen menos trabajo sobre los objetos y tardan un tiempo menor que sus pares que no utilizan el módulo. Una de las causas de esto (además del trabajo con los *Wrappers* mencionado en el *Capítulo 6*) es debido a que tienen menos carga de tratamiento de cadenas de caracteres, ya que el módulo de expresiones encapsula los errores y los reporta de forma optimizada, mejorando así los tiempos finales.

Como posibilidad de mejora sobre este caso de uso se puede mencionar que debido a que existen mecanismos para establecer concurrencia y atomicidad sobre la ejecución de las tareas de curación, se podría proveer al módulo la capacidad de ejecutar expresiones en tareas de forma concurrente, mediante un estudio de la aplicabilidad de estos mecanismos . Esto posibilitaría, por ejemplo, que varias tareas de curación realicen en un mismo momento validaciones sobre un conjunto compartido de elementos. («Curation Task Cookbook - DSpace - DuraSpace Wiki», s. f.)

Trabajos futuros

Existen varias líneas de mejora planteadas, algunas surgidas de la realización de este mismo trabajo y otras planteadas desde el surgimiento de nuevas necesidades, que representan casos de uso distintos para el módulo de expresiones. A continuación se mencionan otras posibilidades de implementación para el módulo de expresiones, a partir de la especificación planteada en el capítulo 5; más adelante se enuncian las mejoras planteadas sobre el módulo de expresiones actual, que surgen como respuesta a nuevas necesidades en el repositorio o a falencias detectadas en el mismo.

Finalmente, a modo de terminación de esta tesina, se enuncian otros casos de uso a implementar, para el módulo de expresiones: mejoras a las tareas de curación, posibilidad de la implementación de un módulo exportador y la definición de un posible sistema de edición por lotes y masificada de objetos del repositorio.

Estudio de otras alternativas de Implementación

En primer lugar, tal y como ya se ha mencionado, el nivel de abstracción con el que se planteó la herramienta de validación en la que se ha separado la definición del lenguaje de su

implementación, resulta muy apropiada para analizar otras implementaciones. Queda para trabajos futuros el estudio de la posibilidad de implementar este lenguaje con otros patrones de implementación para lenguajes específicos de dominio o con otro tipo de soluciones (definidas en el Capítulo 4) para buscar obtener toda la expresividad de la sintaxis abstracta planteada y poderse valer de un lenguaje más sencillo y genérico. De manera concreta, se podría estudiar la posibilidad del desarrollo de un intérprete o un compilador para el lenguaje definido en esta tesina. Se debería utilizar una herramienta más potente y completa que el lenguaje de expresiones definido en la JSR-341 para la implementación.

Una línea de estudio posible es la implementación con la herramienta ANTLR [(«ANTLR 4 Documentation -», s. f.)] de un *DSL externo*, un lenguaje completamente ajeno a cualquier lenguaje de programación actual, que no recibe ningún tipo de limitación en la sintaxis concreta que puede definir. ANTLR es una herramienta que permite definir parsers o intérpretes para un lenguaje personalizado y luego traducir la expresiones del nuevo lenguaje, el lenguaje JAVA (entre otros).

En este contexto, debería implementarse una herramienta, por ejemplo un nuevo módulo de expresiones, que contenga un parser o un intérprete definido en ANTLR, y que se incorpore al módulo *additions*, por lo que pueda ser instanciado desde cualquier punto del software DSpace, para resolver una expresión en el lenguaje planteado. Cabe aclarar que todos los mecanismos de especificación del lenguaje definidos en esta tesina, se podrían reutilizar en la implementación con ANTLR, y el resultado debería ser similar al siguiente:

```
>> Lo que ahora es una expresión del tipo
collection.items().stream().filter(i->!empty(i.getMetadata('dc.type')))
>> podría llegar a ser algo más expresivo como:
SELECTION (ITEMS from Collections) WITH VALIDATION :Item.hasMetadatada(dc.type)
```

Además, el módulo de expresiones se desarrolló con un enfoque incremental y reutilizable, lo que significa que se podrían reutilizar muchos componentes ya construidos, por lo que mejoras de este estilo enfocarse casi exclusivamente el mecanismo utilizado para evaluar las expresiones.

Mejoras al módulo de expresiones

El módulo de expresiones planteado en esta tesina también posee atributos mejorables y características necesarias para resolver problemas que se puedan plantear en el futuro.

Objetos genéricos con identificadores

En el capítulo 6 de esta tesina se menciona la posibilidad de brindar mecanismos que permitan expresiones en el lenguaje que referencien a cualquier objeto a partir de un identificador, sin necesidad de conocer el tipo del objeto sobre el cual se quieren ejecutar las validaciones. Esto sería beneficioso para permitir que las validaciones, selecciones o transformaciones las pueda realizar un usuario administrador con los permisos adecuados, que solo conozca, como ejemplo, un identificador del objeto.

Una posibilidad ya estudiada es que el módulo de expresiones utilice como referencia al objeto sobre el cual se aplica la operación, los identificadores persistentes, del estilo:

```
#handle(10915/30522).getMetadata(dc.type)
#DOI(10.1000/182).getType == 'item'
```

Para poder dar soporte a este tipo de expresiones, ya se comenzó a plantear la mejora del *DSpaceResolver* (ver capítulo 6), para que interprete este tipo de expresiones sin generar cambios sobre la sintaxis concreta del lenguaje, o incluso ampliandola con un aspecto propio del dominio de los repositorios gestionados con DSpace.

Esta mejora será muy provechosa, ya que podría sentar las bases para otro trabajo a realizar en el futuro, del que se hablará más adelante, la posibilidad de que usuarios administradores hagan consultas directas sobre los elementos del repositorio a través de un módulo de exportación o de ejecución por lotes.

Verificación de autorizaciones

Como ya se ha mencionado en el capítulo 6, la expresividad del lenguaje, que ha quedado definida por la sintaxis concreta de la JSR-341, debió limitarse implementando una serie de *Wrappers* de acceso a los objetos de DSpace. Esto se resolvió de este modo porque todavía no se cuenta con mecanismos de autorización para el usuario que controla el módulo de expresiones. En el futuro se brindará la posibilidad de instanciar el módulo con un nivel de autorización, que dependerá de los permisos que tenga el usuario durante la utilización del módulo.

Una idea para concretar este trabajo futuro es que a partir del *EPerson (ver capítulo 2)* que se guarda en el atributo *“Current User” de* la clase *Context* de DSpace, se pueda brindar al módulo soporte para verificar los permisos del usuario cada vez que el usuario instancia y utiliza el módulo. Los permisos que se deben verificar son los gestionados por el módulo de autorización de DSpace; es esta forma, los usuarios con permiso de “READ” sobre los elementos involucrados en la expresión a evaluar podrán utilizar el módulo en modo “*read only*”, mientras que solo los usuarios con permiso “WRITE” podrán operar con el módulo en modo ‘’transformaciones”, y solo aquellos usuarios “ADMIN” de los elementos, podrán utilizar el modo “acceso total” a DSpace. [(DuraSpace, 2012a)]

Implementación de transformaciones

Para realizar distintos trabajos futuros que utilicen el módulo de expresiones, como pueden ser la implementación de nuevas tareas de curación o la utilización en un mecanismo de ejecución por lotes que exprese las transformaciones a realizar en expresiones del lenguaje, es necesario que el módulo de expresiones permita la operación de transformación, que quedó fuera del alcance de esta tesina. Esto requiere desarrollar el segundo nivel de seguridad planteado en el capítulo 6, los Wrappers de transformación, que deben realizar los controles adecuados para asegurar que :

- el usuario tenga permisos para modificar el elemento sobre el que se desarrolla la transformación (con el mecanismo que se explicó en la sección anterior);
- las operaciones de transformación se realicen, mínimamente, de manera atómica. Es decir, ante el surgimiento de algún fallo durante la operación, se debe dar soporte al repositorio para que quede en un estado de consistencia, de acuerdo lo especificado en el estándar ISO correspondiente. [(«ISO/IEC 10026-1:1998», 1998)]

De esta forma, cuando se requiere el módulo de expresiones en modo “transformaciones” o escritura, las operaciones de lectura siguen funcionando igual, pero es posible utilizar los mensajes mutadores de los objetos de DSpace, para realizar cambios en los elementos referenciados. Esto se puede ver en las siguientes expresiones de ejemplo:

- item.setMetadatada(dc.type = ‘Article’) *{cambia el valor del metadato dc.type de un item, al de ‘Article’}*
- bundle.createBitstream(stream) *{crea un bitstream nuevo para el bundle especificado con el stream que se le envía como parámetro}*
- collection.items.stream().forEach(i->i.addBundle(new Bundle(‘*LICENSE*’))) *{Añade a todos los items de una colección determinada, un bundle que viene como parámetro y lo pone como licencia del item}*

Soporte para concurrencia

Con la posibilidad de encapsular el estado del módulo en una clase, queda pendiente el estudio de la viabilidad de una solución capaz de ejecutarse de manera concurrente para la realización de transformaciones sobre los datos. Si bien, la implementación de las transformaciones debe dejar al repositorio en un estado consistente, la posibilidad de que se ejecuten de manera concurrente añade más complejidad y una mayor posibilidad de fallos al módulo. No obstante, existen varios mecanismos aplicables que podrían estudiar en un futuro, sobre todo los descritos en la bibliografía de bases de datos distribuidas, como por ejemplo: fragmentación de datos, mecanismos de replicación, modificación ‘perezosa’ de los datos, módulo de transacciones, etc [(Özsu & Valduriez, 2011)]. Mediante estos mecanismos se podría, por ejemplo, implementar tareas de ejecución diarias y automáticas que recorran los elementos del repositorio, detecten fallos y los puedan corregir utilizando alguna expresión de transformación del lenguaje, sin la necesidad de bloquear el elemento para su sincronización, puesto a que el mismo módulo de expresiones se aseguraría de que el elemento no está siendo accedido en el mismo momento por dos editores distintos y de gestionar los cambios para asegurar la consistencia.

Mejoras en los casos de uso para el módulo de expresiones

Es importante resaltar que las tareas de curación son solo un ejemplo de utilización para el módulo de expresiones. Es posible analizar usos más complejos que explotarán mejor la expresividad del módulo. En esta sección se comienza por listar las mejoras planteadas en un corto plazo para las tareas de curación, pero más adelante se plantea la incorporación de nuevos casos de uso, requeridos por las necesidades que aparecen en el día a día del repositorio institucional.

Tareas de Curación

Una mejora planteada es la parametrización de las expresiones en las tareas que las utilizan, es decir, brindar soporte al administrador del repositorio para que pueda indicar la expresión que la tarea de curación deba evaluar en cada ejecución, sobre cada elemento en el que se aplica. En principio, la implementación de esta mejora no parece costosa si se recurre a algunas características del marco de trabajo de curación propuesto por DSpace. Se podrían utilizar las *taskProperties* [(DuraSpace, s. f.-d)], un mecanismo disponible para cualquier tarea que extienda la clase *AbstractCurationTask,* (tal y como lo hacen las tareas de curación que utilizan el módulo); que permite a las tareas de curación consultar los valores reales de propiedades definidas en archivos de configuración, que pueden variar en tiempo de ejecución. De esta forma, se podría tener un archivo de configuración para la *validationTask,* con el mismo nombre que la tarea de validación (*validationTask.cfg*), donde se podría configurar una propiedad que contenga el valor real de la expresión. Un archivo de configuración posible para esta tarea de curación se puede ver en el *Ejemplo 8.1.*

```
--------------------------------------------------------------------------------
----------Configuraciones utilizadas solamente por --------
----la tarea de validación para el módulo de expresiones---
----------@author Franco Agustín Terruzzi----------
----------------------------------------------------------
validation.ELexpression= {exist(dso.handle)}
----------fin-----------------------------------
```

Ejemplo 1: un archivo de configuración típico de DSpace, pero para la tarea de validación con el módulo de expresiones. Si se cambia la propiedad ELexpression, se puede realizar una nueva validación sobre los elementos en los cuales trabaja la tarea de curación, sin necesidad de modificar ningún código del software DSpace.

De este modo, la tarea de curación se podría ejecutar con la expresión parametrizada en el archivo de configuración, tal y como se puede ver en la *Figura 8.1*

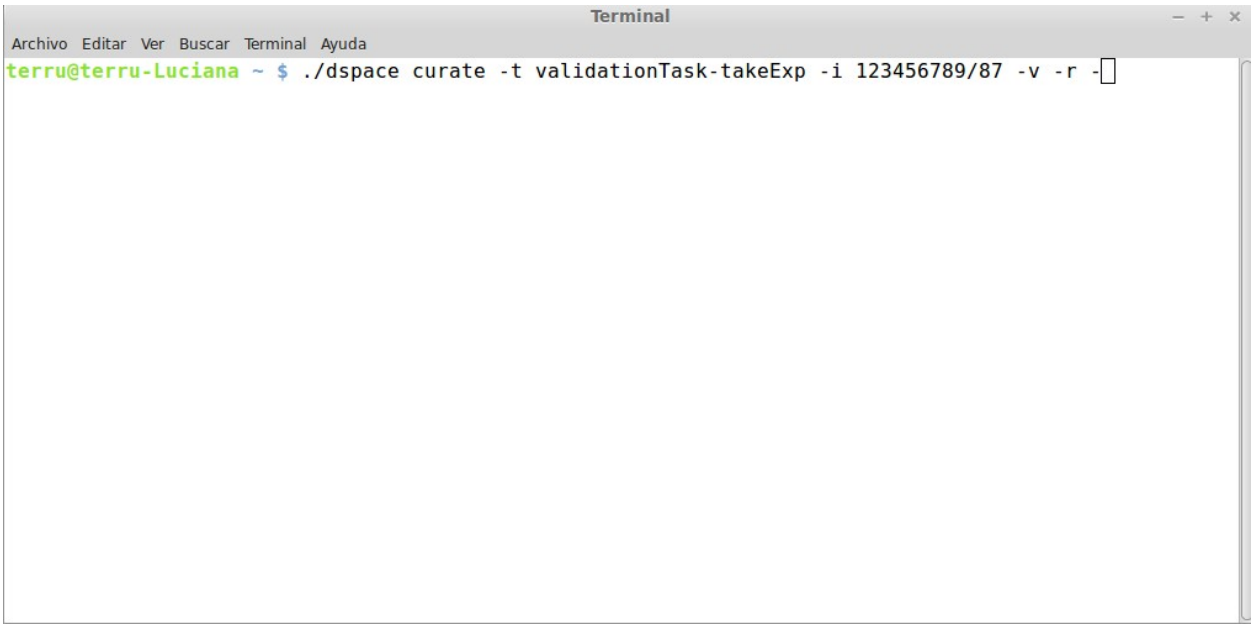

Figura 8.1: una posible ejecución de una tarea de validación con expresión parametrizada

A esta mejora se le suma la de la implementación de las tareas de transformación *transformationTask* y tareas para generar reportes *selectionTask*, que quedan directamente relacionadas con las mejoras ya mencionadas para el módulo de expresiones. Cuando el mismo permita la realización de transformaciones se podrán diseñar tareas de curación que las apliquen; estas tareas se iniciarán desde la misma consola del administrador y permitirán gestionar cambios sobre elementos del repositorio, que podrán ser escritos en expresiones del lenguaje.

Cabe mencionar que las tareas de curación para transformaciones también deberían tener un archivo de configuración para parametrizar las transformaciones. Adicionalmente una *trasnformationTask* podría aplicar una cadena de transformaciones a un elemento, especificada en dicho archivo de configuración y permitir que, por ejemplo, la cadena de transformaciones se ejecute completa o no. La *figura 8.2* muestra como sería la aplicación de una transformación sobre un elemento del repositorio, usando las tareas de curación.

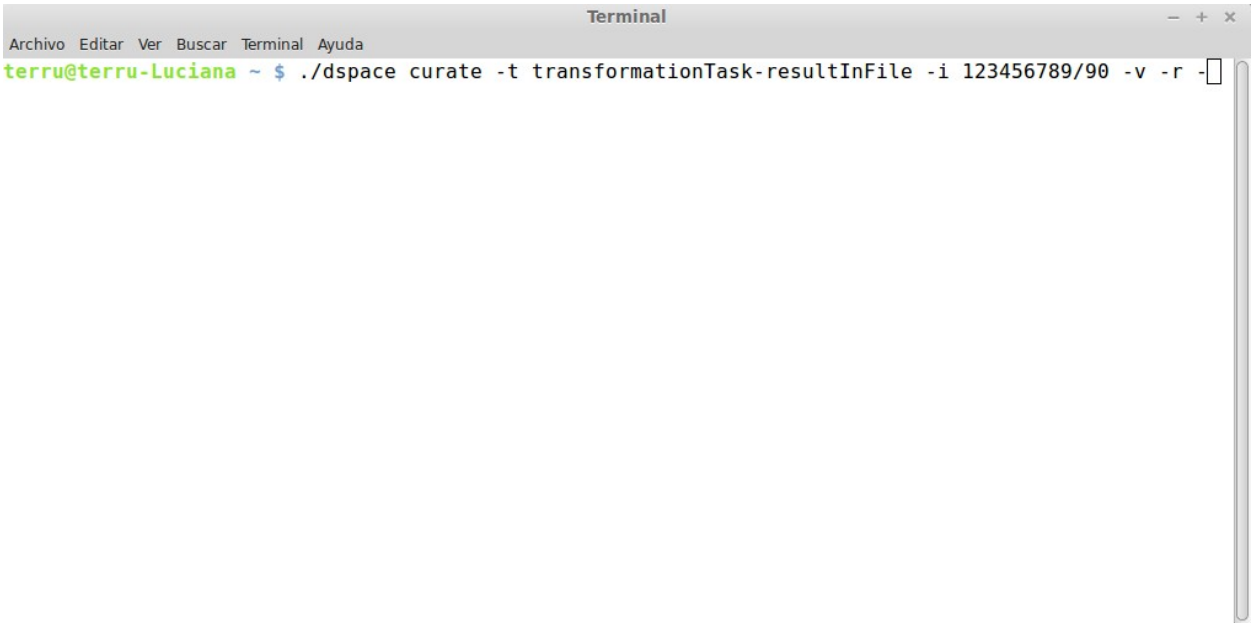

Figura 8.2: posible consola para ejecución de transformaciones. El agregado de la función *resultInFile* permitiría, por ejemplo, guardar los resultados de la aplicación de las transformaciones en un archivo, definido como propiedad de las tareas de curación.

Además de las tareas de curación, existen otros usos posibles para el módulo de expresiones, que se puede instanciar desde cualquier punto de ejecución del software DSpace. El estudio de nuevos casos de uso, depende de los incrementos en la funcionalidad del módulo, a partir de las mejoras planteadas antes en este capítulo. En este caso, ya están planteados dos casos de uso nuevos para el módulo. Los mismos, se describen a continuación.

Módulo de Exportación basado en expresiones

En un futuro cercano, se buscará que el módulo de expresiones se pueda utilizar por el sistema de exportación de Dspace [(DuraSpace, 2012b)]. Se planteará un mecanismo para que los administradores puedan gestionar exportaciones de contenido y metadatos a partir de los resultados de la evaluación de expresión en el lenguaje, probablemente a través de una selección, que permita aislar un conjunto de elementos del repositorio para que se puedan exportar en cualquier de los formatos del módulo de exportación.

Esto sería muy provechoso para los administradores, que podrán contar con mecanismos para construir, por ejemplo, reportes sobre el estado de determinados elementos del repositorio, que cumplan con alguna validación del lenguaje. Entre otras posibilidades, se podría:

- Exportar todos los items que no tengan un bundle ‘Original’
- Exportar los metadatos de un item con un Handle determinado
- Exportar una colección en la cual sus items cumplan alguna condición, por ejemplo no tener algún bitstream.
- Exportar los bitstreams repetidos, detectándolos por su checksum

Se encuentra bajo estudio la implementación de un desarrollo en el módulo additions, que especialice las clases encargadas de la exportación y que se intercomunique con el módulo de expresiones para ejecutar aquellas expresiones que el administrador haya escrito. Luego, el

resultado de la evaluación sería exportado utilizando las herramientas de exportación de DSpace. El panel de control sería algo similar al que se muestra en la *figura 8.3*

Figura 8.3: posible panel de administrador, con el módulo de exportación funcionando y recibiendo las expresiones del módulo de expresión.

Mecanismo de ejecución por lotes

Con la mejora de las transformaciones aplicada sobre el módulo de expresiones, apareceran muchos casos para realizar cambios en el repositorio, a partir de expresiones de transformación. A las tareas de curación ya mencionadas se les puede sumar el desarrollo de un módulo de ejecución por lotes de transformaciones. Este módulo permitiría que se definan ciertos trabajos automáticos (utilizando el sistema de cronjobs de DSpace), que realicen cambios a los elementos, gestionados mediante expresiones de transformación del módulo de expresiones. [(DuraSpace, 2012d)]

El módulo de ejecución por lotes podría añadir una operación al binario de DSpace, que se encargue de evaluar la expresión que le llega como parámetro y guardar el correspondiente resultado en un archivo, a elección del administrador. También, se podría permitir deshacer las transformaciones realizadas, a partir de la aplicación inversa de las mismas, registradas en un '*archivo de transformaciones*'. Además, se podría añadir un cronjob al sistema, que ejecute periódicamente las transformaciones parametrizadas, tal y como lo muestra la figura 8.4, con un archivo crontab de ejemplo.

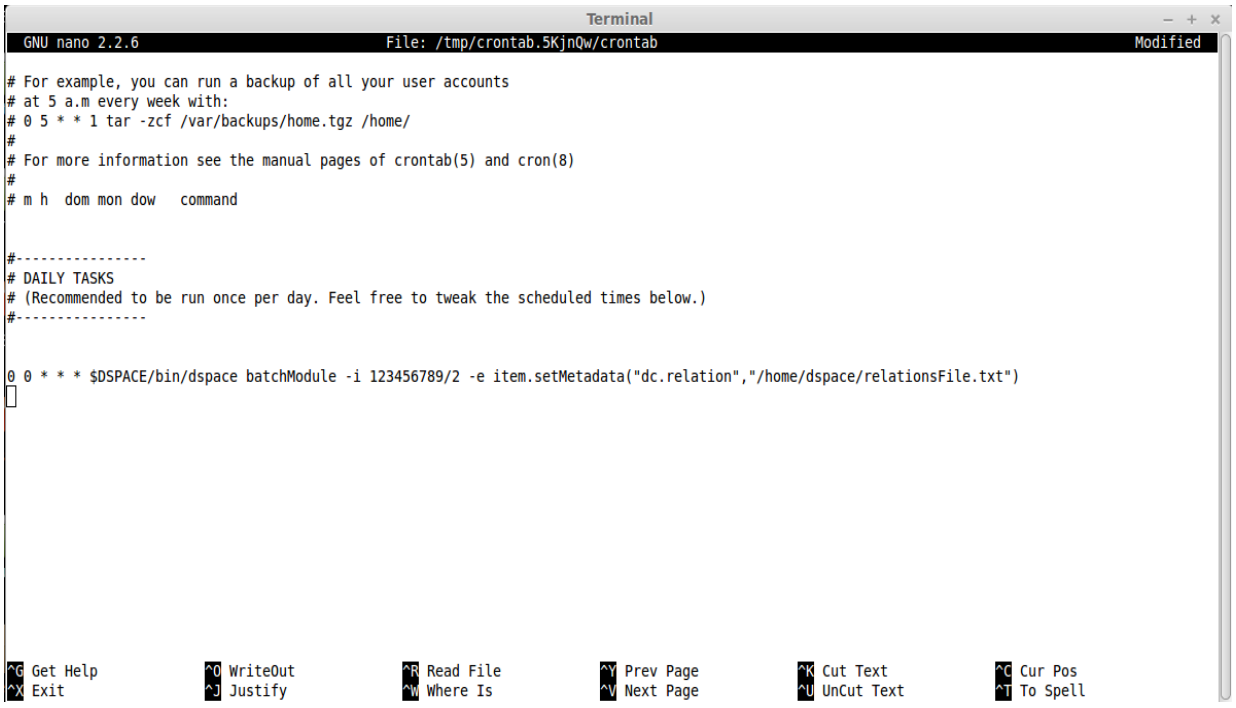

Figura 8.4: un archivo de configuración para tareas automáticas en el servidor que contiene al repositorio. Mediante este tipo de configuraciones, el módulo de ejecución por lotes permitirá realizar transformaciones automáticas a los elementos del repositorio. En este caso, añadir metadatos *relations* a todos los ítems de una colección determinada, donde los valores de las relaciones a añadir, se cargan desde un archivo en el servidor.

Esta funcionalidad será muy útil para los administradores del repositorio, que podrán llevar a cabo tareas de preservación de forma automática, como por ejemplo:

- Agregar un metadato nuevo
- Eliminar un metadato en desuso
- Renombrar un metadato que se haya modificado
- Transformar el valor de un atributo o un metadato a uno o varios elementos
- Normalizar metadatos
- Eliminar elementos repetidos o sin sentido (Ej: colecciones vacías)

Utilización en el Data Provider

Para finalizar esta tesina, se nombra un caso de uso más que queda como posible trabajo futuro: la implementación de un mecanismo de ayuda para el *Data Provider* de OAI [(DuraSpace, 2012c)], para poder configurar conjuntos o sets de OAI, en base a los resultados de la evaluación de ciertas expresiones. Es decir, que se pueda implementar un filtro especial *ExpressionFilter* que permita filtrar a partir de una expresión en el lenguaje planteado. Esto sería muy provechoso, puesto que es más sencillo que el funcionamiento actual, que se realiza a partir de la combinación de filtros simples, donde cada uno valida unos pocos metadatos parametrizados o realiza una validación *ad-hoc* sobre el item. La interoperabilidad es un elemento clave en la gestión de un repositorio, por lo que el estudio de la aplicabilidad del módulo de expresiones a este contexto puede resultar muy fructífero.

Referencias

Akhmechet, S. (2010). Programación funcional para el resto de nosotros, 1-19.

Andino, L. O., & Ruiz, G. E. (2009). Análisis y uso de los frameworks de Eclipse para la definición de DSLs. Recuperado a partir de http://hdl.handle.net/10915/3957

ANTLR 4 Documentation -. (s. f.). Recuperado 25 de marzo de 2015, a partir de https://theantlrguy.atlassian.net/wiki/display/ANTLR4/ANTLR+4+Documentation

Apache Commons. (2013). OGNL - Apache Commons OGNL - Object Graph Navigation Library. Recuperado a partir de http://commons.apache.org/proper/commons-ognl/

Bentley, J. (1986). Programming pearls. *Communications of the ACM*, *29*(8), 711-721. http://doi.org/10.1145/6424.315691

Biblioteca Nacional de Australia. (2003). DIRECTRICES PARA LA PRESERVACIÓN DEL PATRIMONIO DIGITAL. Recuperado 24 de marzo de 2015, a partir de http://unesdoc.unesco.org/images/0013/001300/130071s.pdf

Bootstrap . (s. f.). Recuperado 24 de marzo de 2015, a partir de http://getbootstrap.com/

Bossons, W., Rodgers, R., & Shepherd, K. (2011). Writing and Deploying Your Own Curation Task in DSpace. Recuperado a partir de https://researchspace.auckland.ac.nz/handle/2292/21207

Buffenberger, J., & Gruell, K. (s. f.). A language for software subsystem composition. En *Proceedings of the 34th Annual Hawaii International Conference on System Sciences* (p. 10). IEEE Comput. Soc. http://doi.org/10.1109/HICSS.2001.927267

Cardelli, L., & Davies, R. (1999). Service combinators for Web computing. *IEEE Transactions on Software Engineering*, *25*(3), 309-316. http://doi.org/10.1109/32.798321

Chapter 2. DSpace System Documentation: Functional Overview. (s. f.). Recuperado 19 de junio de 2014, a partir de http://dspace.org/sites/dspace.org/files/archive/1_5_2Documentation/ch02.html#N100A1

Choudhury, G. S. (2008). Case study in data curation at Johns Hopkins University. Recuperado a partir de https://www.ideals.illinois.edu/handle/2142/10669

Chung, K. (2013). JSR-000341 Expression Language 3.0 - Final Release. Recuperado 23 de octubre de 2014, a partir de https://jcp.org/aboutJava/communityprocess/final/jsr341/index.html

Cleaveland, J. C. (1988). Building application generators. *IEEE Software*, *5*(4), 25-33. http://doi.org/10.1109/52.17799

Codehouse. (2009). MVEL - Language Guide for 2.0. Recuperado a partir de http://mvel.codehaus.org/Language+Guide+for+2.0

Consejo Superior de Investigaciones Científicas, C. (s. f.). World | Ranking Web of Repositories. Recuperado 24 de marzo de 2015, a partir de http://repositories.webometrics.info/es/world

Crew, R. F. (1997). ASTLOG: a language for examining abstract syntax trees, 18. Recuperado a partir de http://dl.acm.org/citation.cfm?id=1267950.1267968

Curation Task Cookbook - DSpace - DuraSpace Wiki. (s. f.). Recuperado 18 de junio de 2014, a partir de https://wiki.duraspace.org/display/DSPACE/Curation+Task+Cookbook

De Giusti, M. R. (2010). SeDiCI (Servicio de Difusión de la Creación Intelectual): un recorrido de experiencias (2003-2011). *Jornada Virtual de Acceso Abierto*. Recuperado a partir de http://hdl.handle.net/10915/27160

De Giusti, M. R. (2014a, octubre 1). Las actividades de preservación en un repositorio digital destinadas a dar acceso a lo largo del tiempo a sus contenidos. *XIII Congreso Nacional de Bibliotecología y XX Jornada Nacional y V Internacional de Actualización y Capacitación de Bibliotecas Médicas «Bibliotecas Espacios de Aprendizaje Interdisciplinario» (Bogotá, 2014)*. Recuperado a partir de http://hdl.handle.net/10915/41099

De Giusti, M. R. (2014b, noviembre 28). Una metodología de evaluación de repositorios digitales para asegurar la preservación en el tiempo y el acceso a los contenidos. Recuperado a partir de http://hdl.handle.net/10915/43157

De Giusti, M. R., Lira, A. J., Villarreal, G. L., Terruzzi, F. A., & Adorno, F. G. (2014, marzo 27). Preservación digital: un experimento con SEDICI-DSpace. *XX Asamblea General de ISTEC (Puebla, México, 2014)*. Recuperado a partir de http://hdl.handle.net/10915/34889

De Giusti, M. R., Lira, A. J., Villarreal, G. L., & Texier, J. (2012, noviembre 1). Las actividades y el planeamiento de la preservación en un repositorio institucional. *BIREDIAL - Conferencia Internacional Acceso Abierto, Comunicación Científica y Preservación Digital*. Recuperado a partir de http://hdl.handle.net/10915/26045

De Giusti, M. R., Marmonti, E. H., Sobrado, A., Vila, M. M., & Villarreal, G. L. (2005). Plataforma Celsius - Servicio de Referencia Digital para unidades de información. *III Simposio Internacional de Bibliotecas Digitales (San Pablo, Brasil)*. Recuperado a partir de http://hdl.handle.net/10915/5544

De Giusti, M. R., Oviedo, N. F., Lira, A. J., Sobrado, A., Martínez, J. P., & Pinto, A. V. (2011, mayo 9). SeDiCI - Desafíos y experiencias en la vida de un repositorio digital. *I Conferencia sobre Bibliotecas y Repositorios Digitales (BIREDIAL) (Colombia, 2011)*. Recuperado a partir de http://hdl.handle.net/10915/5528

De Giusti, M. R., Oviedo, N. F., Lira, A. J., & Villarreal, G. L. (2013, octubre 17). Control de integridad y calidad en repositorios DSpace. *III Conferencia de Bibliotecas y Repositorios Digitales de América Latina (BIREDIAL) y VIII Simposio Internacional de Bibliotecas Digitales (SIBD) (Costa Rica, 2013)*. Recuperado a partir de http://hdl.handle.net/10915/30524

De Giusti, M. R., Sobrado, A., Lira, A. J., Vila, M. M., & Villarreal, G. L. (2008). SeDiCI | Servicio de Difusión de la Creación Intelectual - UNLP. *Revista Interamericana de Bibliotecología*. Recuperado a partir de http://hdl.handle.net/10915/5524

Deursen, A. Van, Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *ACM Sigplan Notices*, *35*(6), 26-36. http://doi.org/10.1145/352029.352035

Dissertations, N. D. L. of T. and. (s. f.). NTLTD catalog. Recuperado 24 de marzo de 2015, a partir de http://www.ndltd.org/

Doxigen. (s. f.). dspace: ItemIterator.java Source File. Recuperado 25 de marzo de 2015, a partir de http://fossies.org/dox/dspace-4.2-src-release/ItemIterator_8java_source.html

Dublin core, M. I. (s. f.). DCMI Specifications. Recuperado 24 de marzo de 2015, a partir de http://dublincore.org/specifications/

DuraSpace. (s. f.-a). Advanced Customisation - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Advanced+Customisation

DuraSpace. (s. f.-b). Architecture - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Architecture

DuraSpace. (s. f.-c). Business Logic Layer - DSpace 5.x Documentation. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Business+Logic+Layer#BusinessLogicLayer-TheConfigurationManager

DuraSpace. (s. f.-d). Curation System - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Curation+System

DuraSpace. (s. f.-e). Discovery - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado a partir de https://wiki.duraspace.org/display/DSDOC5x/Discovery

DuraSpace. (s. f.-f). DSpace 5.x Documentation - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado a partir de https://wiki.duraspace.org/display/DSDOC5x/DSpace+5.x+Documentation

DuraSpace. (s. f.-g). DSpaceContributors - DSpace - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSPACE/DspaceContributors

DuraSpace. (s. f.-h). PluginManager - DSpace - DuraSpace Wiki. Recuperado 25 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSPACE/PluginManager

DuraSpace. (s. f.-i). Release Notes - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Release+Notes

DuraSpace. (s. f.-j). REST API - DSpace 4.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC4x/REST+API

DuraSpace. (s. f.-k). RoadMap - DSpace - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSPACE/RoadMap

DuraSpace. (s. f.-l). Search Engine Optimization - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 24 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Search+Engine+Optimization

DuraSpace. (2012a). Functional Overview - DSpace 5.x Documentation - DuraSpace Wiki. Recuperado 25 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Functional+Overview#FunctionalOverview-Authorization

DuraSpace. (2012b). Ingesting Content and Metadata - DSpace 5.x Documentation. Recuperado 25 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Ingesting+Content+and+Metadata

DuraSpace. (2012c). OAI-PMH Data Provider 2.0 (Internals) - DSpace 5.x Documentation. Recuperado 25 de marzo de 2015, a partir de https://wiki.duraspace.org/pages/viewpage.action?pageId=45548245

DuraSpace. (2012d). Scheduled Tasks via Cron - DSpace 5.x Documentation -. Recuperado 25 de marzo de 2015, a partir de https://wiki.duraspace.org/display/DSDOC5x/Scheduled+Tasks+via+Cron

Duval, E., Hodgins, W., Sutton, S., & Weibel, S. L. (s. f.). Metadata Principles and Practicalities. Recuperado 24 de marzo de 2015, a partir de http://www.dlib.org/dlib/april02/weibel/04weibel.html?utm_source=dbpia&utm_medium=article_detail&utm_campaign=reference

Fedora Repository | Fedora is a general-purpose, open-source digital object repository system. (s. f.). Recuperado 24 de marzo de 2015, a partir de http://fedora-commons.org/

Fowler, M. (2005). Language Workbenches: The Killer-App for Domain Specific Languages? Recuperado 24 de octubre de 2014, a partir de http://martinfowler.com/articles/languageWorkbench.html

Fowler, M. (2010). *Domain-Specific Languages*. Recuperado a partir de https://books.google.com/books?hl=es&lr=&id=ri1muolw_YwC&pgis=1

Fugueras, R. A. i, Graells, M. T., García, A. C., Rodríguez, R. Á., García, A. del A., Flecha., E. M., & Brinquis, M. del C. H. (2011). Jornadas Archivando: La preservación en los archivos. Recuperado 24 de marzo de 2015, a partir de https://archivofsierrapambley.files.wordpress.com/2011/12/actas_archivando_2011.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Recuperado a partir de http://dl.acm.org/citation.cfm?id=186897

GC: ExpressionFactoryImpl . (s. f.). Recuperado 25 de marzo de 2015, a partir de http://grepcode.com/file/repo1.maven.org/maven2/org.glassfish.web/javax.el/2.2.2/com/sun/el/ExpressionFactoryImpl.java

Germon, R. (s. f.). Using xml as an intermediate form for compiler development. Recuperado a partir de http://www.researchgate.net/publication/238674289_Using_xml_as_an_intermediate_form_for_compiler_development

Ghezzi, C., & Jazayeri, M. (1986). Conceptos de lenguajes de programación. Díaz de Santos. Recuperado a partir de http://dialnet.unirioja.es/servlet/libro?codigo=139533

Gondow, K., & Kawashima, H. (2002). Towards ANSI C Program Slicing using XML. *Electronic Notes in Theoretical Computer Science*, *65*(3), 30-49. http://doi.org/10.1016/S1571-0661(04)80425-0

Gray, J., & Karsai, G. (2003). An examination of DSLs for concisely representing model traversals and transformations. En *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the* (p. 10 pp.). IEEE. http://doi.org/10.1109/HICSS.2003.1174892

Hanus, M. (2007). Logic Programming. En V. Dahl & I. Niemelä (Eds.), (Vol. 4670). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-74610-2

Information, N. C. for B. (s. f.). National Center for Biotechnology Information. Recuperado 24 de marzo de 2015, a partir de http://www.ncbi.nlm.nih.gov/

ISO/IEC 10026-1:1998. (1998). Recuperado 21 de marzo de 2015, a partir de https://www.iso.org/obp/ui/#iso:std:iso-iec:10026:-1:ed-2:v1:en

Java Community Process. (s. f.). The Java Community Process(SM) Program. Recuperado 25 de marzo de 2015, a partir de https://www.jcp.org/en/home/index

java.util (Java Platform SE 7). (s. f.). Recuperado 25 de marzo de 2015, a partir de http://docs.oracle.com/javase/7/docs/api/java/util/package-summary.html

Jones, J. (2003). Abstract Syntax Tree Implementation Idioms. *Proceedings of the 10th Conference on Pattern Languages of Programs (PLoP2003)*, 1-10. Recuperado a partir de http://www.hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf

Juneau, J. (2013). *Introducing Java EE 7: A Look at What's New*. Apress. Recuperado a partir de https://books.google.com/books?id=Cd1o9M8G_nkC&pgis=1

Kamin, S. N., & Hyatt, D. (1997). A Special-Purpose Language for Picture-Drawing. *Proceedings of the Conference on Domain-Specific Languages, October 15--17, 1997, Santa Barbara, California*, (October), 297-310. Recuperado a partir de http://www.usenix.org/publications/library/proceedings/dsl97/full_papers/kamin/kamin_html/kamin.html

Kimpton, M. (s. f.). DSpace Foundation aims to provide expanded access to scholarly works. Recuperado 24 de marzo de 2015, a partir de http://www.hpl.hp.com/news/2007/oct-dec/dspace_foundation.html

Kiselyov, O., & Lisovsky, K. (2002). XML, XPath, XSLT implementations as SXML, SXPath, and SXSLT. *International Lisp Conference, ILC'2002*.

Koutsomitropoulos, D. A., Solomou, G. D., Papatheodorou, T. S. P., & Alexopoulos, A. D. (2010). The Use of Metadata for Educational Resources in Digital Repositories: Practices and Perspectives. *D-Lib Magazine*. Corporation for National Research Initiatives. Recuperado a partir de http://dialnet.unirioja.es/servlet/citart?info=link&codigo=3742624&orden=318436

Kurtz, M. (2013). Dublin Core, DSpace, and a Brief Analysis of Three University Repositories. *Information Technology and Libraries*, *29*(1), 40-46. http://doi.org/10.6017/ital.v29i1.3157

LA Referencia |. (2012, marzo 5). Recuperado a partir de http://lareferencia.redclara.net/rfr/

Lagoze, C., Van de Sompel, H., Nelson, M., & Warner, S. (2015). Open Archives Initiative - Protocol for Metadata Harvesting - v.2.0. Recuperado a partir de http://www.openarchives.org/OAI/openarchivesprotocol.html

Latin America | Ranking Web of Repositories. (s. f.). Recuperado 24 de marzo de 2015, a partir de http://repositories.webometrics.info/es/Latin_America

Launchbury, J., Lewis, J. R., & Cook, B. (1999). On embedding a microarchitectural design language within Haskell. En *Proceedings of the fourth ACM SIGPLAN international conference on Functional programming - ICFP '99* (Vol. 34, pp. 60-69). New York, New York, USA: ACM Press. http://doi.org/10.1145/317636.317784

Louden, K. C. (2004). *Construcción de compiladores: principios y práctica*. Thomson. Recuperado a partir de http://books.google.com.ar/books/about/Construcci%C3%B3n_de_compiladores.html?id=hLVAOIb2FhsC&pgis=1

Madaan, A., & Bhalla, S. (2013). Domain specific multistage query language for medical document repositories. *Proceedings of the VLDB Endowment*, *6*(12), 1410-1415. http://doi.org/10.14778/2536274.2536327

Manual, R. (2013). GlassFish Reference Manual Release 4.0.

Massol, V., & Husted, T. (2003). JUnit in Action. Recuperado a partir de http://dl.acm.org/citation.cfm?id=961868

Mernik, M., Heering, J., & Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, *37*(4), 316-344. http://doi.org/10.1145/1118890.1118892

Moreira, M. A. (1997). El conocimiento humano es construido; el aprendizaje significativo subyace a esa construcción. (J.D. Novak)., (1997), 19-44.

Nancy, C. (2010). *More Technology for the Rest of Us: A Second Primer on Computing for the Non-IT Librarian*. Recuperado a partir de https://books.google.com/books?hl=es&lr=&id=kQpz3MXjM_8C&pgis=1

Omg. (2011). UML Infrastructure Specification, v2.4.1. *Omg*, (August), 34. Recuperado a partir de http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/

OpenDOAR Chart - Growth of the OpenDOAR Database - Worldwide. (s. f.). Recuperado 2 de julio de 2014, a partir de http://www.opendoar.org/onechart.php?cID=&ctID=&rtID=&clID=&lID=&potID=&rSoftWareName=&search=&groupby=r.rDateAdde

d&orderby=&charttype=growth&width=600&height=350&caption=Growth of the OpenDOAR Database - Worldwide

Oracle. (s. f.-a). Arrays. Recuperado 25 de marzo de 2015, a partir de https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Oracle. (s. f.-b). Chapter 15. Expressions. Recuperado 25 de marzo de 2015, a partir de http://docs.oracle.com/javase/specs/jls/se7/html/jls-15.html#jls-15.16

Oracle. (s. f.-c). RuntimeException. Recuperado 25 de marzo de 2015, a partir de http://docs.oracle.com/javase/7/docs/api/java/lang/RuntimeException.html

Oracle. (s. f.-d). Summary of Operators. Recuperado 25 de marzo de 2015, a partir de https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

Oracle. (1999). Code Conventions for the Java Programming Language: 9. Naming Conventions. Recuperado 25 de marzo de 2015, a partir de http://www.oracle.com/technetwork/java/codeconventions-135099.html

Özsu, M. T., & Valduriez, P. (2011). *Principles of Distributed Database Systems*. *Management* (Vol. 12). http://doi.org/10.1007/978-1-4419-8834-8

Parigot, D., & Courbis, C. (2005). Domain-Driven Development: the SmartTools Software Factory. *Sophia*, 19. http://doi.org/10.1145/1028664.1028685

Parks, C., Nelson, S., & Mitra. (s. f.). The Model Primary Content Type for Multipurpose Internet Mail Extensions. Recuperado a partir de http://tools.ietf.org/html/rfc2077

Perrin, J. M., Winkler, H. M., & Yang, L. (2015). Digital Preservation Challenges with an ETD Collection — A Case Study at Texas Tech University. *The Journal of Academic Librarianship*, *41*(1), 98-104. http://doi.org/10.1016/j.acalib.2014.11.002

PREMIS Data Dictionary for Preservation Metada. (s. f.). Recuperado 8 de octubre de 2014, a partir de http://www.loc.gov/standards/premis/v2/premis-2-2.pdf

Primitive Data Types. (s. f.). Recuperado 25 de marzo de 2015, a partir de https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Process, J. C. (s. f.). Maven Repository: javax.el » el-api. Recuperado 25 de marzo de 2015, a partir de http://mvnrepository.com/artifact/javax.el/el-api

Productiva, M. de C. y T. e I. Directrices SNRD Directrices para proveedores de contenido del Sistema Nacional de Repositorios Digitales Ministerio de Ciencia, Tecnología e Innovación Productiva. Recuperado a partir de http://repositorios.mincyt.gob.ar/pdfs/Directrices_SNRD_2012.pdf

Productiva, M. de C. y T. e I. Plan de actividades para la creación y el fortalecimiento del SNRD Bases para la Solicitud de Apoyo Económico. Recuperado a partir de http://repositorios.mincyt.gob.ar/documentos/fortalec_bd/fortalec_bd_bases_SNRD.pdf

Productiva, M. de C. y T. e I. Repositorios DRIVER. Recuperado a partir de http://repositoriosdigitales.mincyt.gob.ar:8380/dnet-web-generic/showRepositories.action

Productiva, M. de C. y T. e I. Resolución 469/11. Recuperado a partir de http://www.mincyt.gob.ar/adjuntos/archivos/000/021/0000021632.pdf

Productiva, M. de C. y T. e I. (s. f.-e). Sistema Nacional de Repositorios Digitales - República Argentina. Recuperado 24 de marzo de 2015, a partir de http://repositorios.mincyt.gob.ar/index.php

Rauber, A., Christodoulakis, S., & Tjoa, A. M. (Eds.). (2005). *Research and Advanced Technology for Digital Libraries* (Vol. 3652). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/11551362

Reference Model for an Open Archival Information System (OAIS), CCSDS Magenta Book. Issue 1. . (s. f.). Recuperado 28 de mayo de 2014, a partir de http://public.ccsds.org/publications/archive/650x0m2.pdf

Sanaulla, M. (2013). Creating Internal DSLs in Java, Java 8- Adopting Martin Fowler's approach | Javalobby. Recuperado 25 de marzo de 2015, a partir de http://java.dzone.com/articles/creating-internal-dsls-java

Seamframework. (2009). Chapter 30. JBoss EL. Recuperado a partir de http://docs.jboss.org/seam/2.0.2.SP1/reference/en-US/html/elenhancements.html

Silva, J. R. da, Ribeiro, C., & Lopes, J. C. (2011). A Data Curation Experiment at U.Porto using DSpace. Recuperado 25 de marzo de 2015, a partir de http://dendro.fe.up.pt/pdf/papers/ipres2011.pdf

Simos, M. A. (1995). Organization domain modeling (ODM). *ACM SIGSOFT Software Engineering Notes*, *20*(SI), 196-205. http://doi.org/10.1145/223427.211845

Sirer, E. G., & Bershad, B. N. (2000). Using production grammars in software testing. *ACM SIGPLAN Notices*, *35*(1), 1-13. http://doi.org/10.1145/331963.331965

Smaragdakis, Y., & Batory, D. (1997). DiSTiL: A transformation library for data structures. *USENIX Conference on Domain-Specific Languages*, (October), 257270.

Smith, M., Barton, M., Branschofsky, M., McClellan, G., Walker, J. H., Bass, M., … Tansley, R. (2003). DSpace. *D-Lib Magazine*, *9*(1). http://doi.org/10.1045/january2003-smith

Srinivasan, K. (2006). Unified Expression Language for JSP and JSF. Recuperado 25 de marzo de 2015, a partir de https://today.java.net/pub/a/today/2006/03/07/unified-jsp-jsf-expression-language.html

Steinhart, G., & Lowe, B. (2007, abril 1). Data Curation and Distribution in Support of Cornell University's Upper Susquehanna Agricultural Ecology Program. DigCCurr2007 Proceedings: DigCCurr2007, an International Symposium on Digital Curation. Recuperado a partir de http://ecommons.library.cornell.edu/handle/1813/7517

Stevenson, D. E., & Fleck, M. M. (s. f.). Programming Language Support for Digitized Images or, The Monsters in the Closet. Recuperado 25 de marzo de 2015, a partir de https://www.usenix.org/legacy/publications/library/proceedings/dsl97/full_papers/stevenson/stevenson_html/stevenson.html

Strodl, S., Becker, C., Neumayer, R., & Rauber, A. (2007). How to choose a digital preservation strategy. En *Proceedings of the 2007 conference on Digital libraries - JCDL '07* (p. 29). New York, New York, USA: ACM Press. http://doi.org/10.1145/1255175.1255181

Sun, S., Reilly, S., Lannom, L., & Petrone, J. (s. f.). RFC 3652-Handle System Protocol (v2.1)-November 2003. Recuperado 19 de junio de 2014, a partir de http://www.ietf.org/rfc/rfc3652.txt

SWORD V2 Specifications . (s. f.). Recuperado 24 de marzo de 2015, a partir de http://swordapp.org/sword-v2/sword-v2-specifications/

Térmens, M. (2009). Investigación y desarroiio en preservación digitai: un baiance internacional. Recuperado 24 de marzo de 2015, a partir de http://tic.uis.edu.co/ava/pluginfile.php/152259/mod_resource/content/2/TERMENS, MIQUEL. INVESTIGACI%C3%93N Y DESARROLLO EN PRESERVACI%C3%93N DIGITAL.pdf

Térmens, M. (2013). *Preservación digital: Editorial UOC, Niberta*. Recuperado a partir de http://www.editorialuoc.cat/preservacindigital-p-1167.html?cPath=1http://tic.uis.edu.co/ava/pluginfile.php/152259/mod_resource/content/2/TERMENS, MIQUEL. INVESTIGACI%C3%93N Y DESARROLLO EN PRESERVACI%C3%93N DIGITAL.pdf

Terruzzi, F. A., Lira, A. J., Villarreal, G. L., & De Giusti, M. R. (2014, octubre 1). Evaluación automática de preservación mediante la utilización de tareas de curación en DSpace. *Conferência Internacional Acesso Aberto, Preservação Digital, Interoperabilidade, Visibilidade e Dados Científicos - BIREDIAL 2014 (Brasil, 2014)*. Recuperado a partir de http://hdl.handle.net/10915/44573

Texier, J., De Giusti, M. R., & Gordillo, S. E. (2014). El desarrollo de software dirigido por modelos en los repositorios institucionales. *DYNA*. Recuperado a partir de http://hdl.handle.net/10915/35601

Tucker, A. (2003). Lenguajes de programación : principios y paradigmas. Recuperado 25 de marzo de 2015, a partir de http://www.sidalc.net/cgi-bin/wxis.exe/?IsisScript=AGRIUAN.xis&method=post&formato=2&cantidad=1&expresion=mfn=029241

Value and Method Expressions . (s. f.). Recuperado 25 de marzo de 2015, a partir de http://docs.oracle.com/javaee/6/tutorial/doc/bnahu.html

Van Deursen, A., & Klint, P. (2002). Domain-Specific Language Design Requires Feature Descriptions. *Journal of Computing and Information Technology*, *10*(1), 1-17. http://doi.org/10.2498/cit.2002.01.01

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages. *ACM SIGPLAN Notices*, *35*(6), 26-36. http://doi.org/10.1145/352029.352035

Van Engelen, R. a. (2001). ATMOL: A Domain-Specific Language for Atmospheric Modeling. *Journal of Computing and Information Technology*, *9*(4), 289-303. http://doi.org/10.2498/cit.2001.04.02

Voelter, M. (2013). DSL Engineering: Designing, Implementing and Using Domain-Specific Languages: Markus Voelter: 9781481218580: Amazon.com: Books. Recuperado 27 de octubre de 2014, a partir de http://www.amazon.com/DSL-Engineering-Designing-Implementing-Domain-Specific/dp/1481218581

Westra, B., Ramirez, M., Parham, S. W., & Scaramozzino, J. M. (2010). Selected Internet Resources on Digital Research Data Curation. *Issues in Science and Technology Librarianship*. Association of College and Research Libraries. Recuperado a partir de http://dialnet.unirioja.es/servlet/citart?info=link&codigo=3706831&orden=308481