INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

SEARCH
TECHNOLOGIES

SEARCH TECHNOLOGIES LATAM S.A.

"MIGRACIÓN DE PROCESOS DE CONVERSIÓN EN PERL A ASPIRE"

INFORME DE PROYECTO DE GRADUACIÓN PARA OPTAR POR EL GRADO DE BACHILLER EN INGENIERÍA EN COMPUTACIÓN

LUIS JAVIER MÉNDEZ VÁSQUEZ

CARTAGO, NOVIEMBRE 2010

RESUMEN

Este proyecto tiene como objetivo analizar una serie de procesos realizados para un cliente de la empresa Search Technologies. Estos procesos transforman documentos electrónicos a un formato más estructurado que permita su posterior indexación y publicación, ya sea tanto en línea como fuera de línea por medio de herramientas como Rocket NXT y Folio. Una vez analizados estos procesos, se han migrado a Aspire, una plataforma para procesamiento de documentos de alto rendimiento perteneciente a Search Technologies.

Los procesos escritos en Perl originalmente tienen como entrada documentos en formato Microsoft Word 2003, PDF y HTML. Estos documentos deben ser convertidos a formato HTML si no lo estuvieran y luego recibir una serie de transformaciones textuales mediante el uso de expresiones regulares, con el fin de que su posterior indexación y publicación pueda ser llevada a cabo satisfactoriamente. Este proceso es actualmente semiautomático, es decir, una parte del proceso debe ser realizada manualmente. Este proyecto fue capaz de automatizar el proceso en su totalidad y facilitar futuras implementaciones para otros clientes con necesidades similares.

La migración realizada consiste en la creación de pipelines de procesamiento en un ambiente de ejecución concurrente de muchos hilos y el uso extensivo de expresiones regulares.

Palabras claves

Proceso; Conversión; Expresiones Regulares; Perl; Java; Migración; Concurrencia; Pipeline de Procesamiento; Transformación; Extracción de Metadatos.

ABSTRACT

The goal of this project is to analyze a series of processes done for a particular client of Search Technologies Company. Those processes transform electronic documents to a more structured format that will allow them to be indexed and published online or offline using Rocket NXT and Folio products. Once analyzed, those processes have been migrated to Aspire, a high performance document processing framework property of Search Technologies.

The processes originally written in Perl have as input Microsoft Word 2003 documents, PDF and HTML files. Those documents must be converted to HTML format, if they were not yet, and then passed through a series of text transformations using regular expressions, with the intention of a later successful indexation and publication. This process is semiautomatic, i.e., part of the process is done manually. This project was capable of automate the process entirely and facilitate future implementations for other clients with similar needs.

The migration consists of the creation of processing pipelines in a multithread concurrent environment and the extensive use of regular expressions.

Keywords

Process; Conversion; Regular Expressions; Perl; Java; Migration; Concurrency; Process pipeline; Transformation; Metadata extraction.

Índice General

Capítulo 1
Descripción del Problema

La siguiente sección da una breve reseña de la empresa en que se realizó el proyecto de práctica, Search Technologies. Asimismo describe los antecedentes del proyecto y el contexto en que surge. Se da una descripción general de las necesidades del cliente y cómo el actual proyecto de práctica ha podido resolverlas. Adicionalmente, se proporciona una lista de las personas que participaron en el proyecto ya sea como consultores o como administradores de proyecto.

1.1 Contexto del Proyecto

Search Technologies es una compañía internacional, con sede principal en Virginia, Estados Unidos. Ésta empresa es proveedora independiente de experticia en motores de búsqueda y servicios de integración. Estos servicios permiten a sus clientes obtener ventaja competitiva en el ámbito de negocios utilizando búsqueda. La consultoría ofrecida por esta compañía en selección de productos permite llevar implementaciones eficientes y efectivas además de soporte para aplicaciones de búsqueda.

A finales del año 2009, Search Technologies adquirió una compañía llamada InfoSolutions. Esta empresa se dedicaba a proveer soluciones de búsqueda a gran variedad de clientes usando productos de la línea Rocket NXT y Folio (Suites de publicación). En el momento de la adquisición, los clientes y contratos de InfoSolutions pasan a ser de Search Technologies. Algunos de estos clientes mantienen contratos de mantenimiento y soporte, puesto que la implementación de una solución de búsqueda ya fue realizada para ellos. Por esta razón es necesaria la constante actualización de los contenidos que estos clientes han publicado para sus lectores, con el fin de que la información provista sea la más exacta, actualizada y confiable.

Actualmente la nueva oficina de Search Technologies, ubicada en Kentucky (antes InfoSolutions) se encarga de actualizar el contenido publicado para esta agenda de clientes. El proceso de recibir el contenido original, migrarlo a un formato más flexible, indexarlo y combinarlo con el contenido ya publicado es hecho manualmente con una combinación de Scripts escritos en Perl.

La experiencia en motores de búsqueda y metodologías que ha adquirido Search Technologies tras muchas implementaciones exitosas en el negocio de búsquedas, ha permitido la creación de una herramienta especializada en el procesamiento de documentos. Ésta plataforma, desarrollada en Java, permite automatizar procedimientos similares a los que actualmente son hechos por la oficina de Kentucky. Es por esta razón que, aunque dichos procesos generados por la antigua InfoSolutions se han refinado y funcionan correctamente, éstos serán migrados a Aspire, la plataforma para procesamiento de documentos de alto rendimiento propietaria de Search Technologies.

La razón tras esta decisión, yace en el hecho de que la misma hará más robusto a Aspire, al agregar nuevos módulos que estarán disponibles para futuras implementaciones. Sin embargo la razón principal de esta migración es que el proceso manual existente pasará a ser automático, lo cual ahorrará tiempo para el personal de Search Technologies incrementando así las ganancias de la compañía y mejorando el servicio al cliente.

Este proyecto en particular, abarca a un cliente en específico. AILA (American Immigration Lawyers Association), es una asociación estadounidense de abogados y profesores de leyes que practican y enseñan ley de inmigración. AILA necesita publicar gran cantidad de noticias, leyes, artículos, libros y otro tipo de documentos utilizando la web. Además, requiere llevar exactamente el mismo contenido a sus miembros a través de otros medios, por ejemplo discos compactos.

Este proyecto de práctica consistió en tomar el proceso de actualización de contenidos para AILA y migrarlo a Aspire. Se necesitó investigar y diseñar una solución que fuera fácil de mantener, reusable y eficiente para que futuras migraciones sean posibles en el menor tiempo posible. El proyecto fue realizado para el Departamento de Soporte de Search Technologies, sede de Costa Rica.

1.2 Descripción del Problema

Los distintos clientes necesitan hacer actualizaciones al contenido que han publicado periódicamente (por ejemplo cada mes o trimestre) o en algunos casos, cuando ellos lo decidan. Esta actualización tiene que ser eficiente y efectiva, de tal manera que los cambios realizados en la colección de documentos se vean reflejados en el contenido publicado. Actualmente este proceso es realizado manualmente, lo que conlleva a errores y atrasos innecesarios. Esto afecta principalmente a los clientes y personal de Search Technologies, quienes se ven en la necesidad de gastar tiempo en el proceso de actualización en lugar de continuar con otras tareas de mayor prioridad.

La solución propuesta es migrar el proceso de actualización a Aspire y transformarlo en un procedimiento automático que únicamente requiera supervisión humana para aseguramiento de la calidad (revisión del contenido actualizado). Esta solución deberá permitir a los clientes realizar actualizaciones con la misma frecuencia que ahora, pero en menor tiempo (ya sea de reparación de errores o espera) para ellos y el personal de Search Technologies. El desarrollo propuesto no debe representar un cambio mayor al método utilizado por el cliente actualmente para realizar la actualización, con el propósito de facilitar la aceptación del mismo.

1.3Objetivos y Alcance

1.3.1 General

a. Migrar el proceso manual existente de actualización de contenido para AILA a la plataforma para el procesamiento de documentos Aspire.

1.3.2 Específico

a. Analizar el proceso actual hecho manualmente y scripts para comprender las necesidades del cliente.
b. Proponer y diseñar una alternativa al proceso utilizando la plataforma Aspire.
c. Desarrollar y documentar la alternativa propuesta.
d. Implementar la alternativa en el servidor de procesamiento utilizado por los clientes.

1.4 Involucrados

A continuación se muestra una lista con la información de las personas involucradas en el proyecto:

Tabla 1.1 Personal involucrado en el proyecto

Nombre	Posición	Rol
Maynor Alvarado	Jefe de Operaciones en Costa Rica	Coordinador desde Costa Rica
Bob Berberich	Jefe de Operaciones en Kentucky	Soporte y Coordinación desde Kentucky
Jonathan González	Desarrollador	Consultor en Aspire
Luis Luna	Desarrollador	Consultor en Aspire
Steve Miller	Desarrollador	Consultor y entrenador en los procesos de Perl
Tim Naylor	Desarrollador	Consultor y entrenador en NXT y Folio.
Luis Pintor	Administrador del Departamento de soporte	Administrador del proyecto
Stan Weldy	Desarrollador	Consultor en los procesos de Perl.

Capítulo 2

Proceso de Actualización de Contenidos

A continuación se detalla el proceso de actualización de contenidos realizado por AILA.

2.1 Descripción General

AILA publica contenido utilizando Rocket NXT Online Server y Solo Offline Publisher. El primero, permite publicar contenido a través de Internet; mientras que el segundo habilita al usuario para crear una versión empaquetada de los contenidos que una vez adquirida por el usuario, éste instala una versión local de la colección de documentos en su computador. Las actualizaciones a dicha colección son realizadas mediante archivos independientes y secuenciales, que son descargados automáticamente por el asistente de instalación provisto junto con la colección de documentos.

El proceso de actualización recibe archivos en formato Microsoft Word 2003 (.doc) y PDFs (.pdf), los combina y transforma en un único archivo de colección propio de Rocket NXT (.nxt) o los resume en un archivo de actualización (.upd) para posteriormente enviarlos al servidor de producción donde los clientes podrán acceder al contenido actualizado. Cuando AILA requiere actualizar la colección, el administrador de contenido[1] accede al directorio FTP llamado *root*. Cualquier cambio en dicho directorio, será reflejado en la tabla de contenidos de la colección publicada.

La primera fase del proceso es realizada por un script escrito en Perl. Éste script detecta cuales documentos deben ser procesados, los transforma a HTML (si son documentos de Word 2003), aplica algunos arreglos sobre los documentos ya transformados y combinarlos con la colección existente para su posterior publicación en línea.

[1] Persona encargada de administrar el contenido de la colección. Se encarga de editar, borrar, mover o agregar archivos a la colección. Además se encarga de iniciar el proceso de actualización y reparar errores en el mismo.

Una vez la colección en línea se ha construido, el personal de Search Technologies deberá realizar tareas adicionales para tener la actualización del contenido sin conexión. Este trabajo adicional consiste en construir los archivos de actualización y combinarlos con la colección existente sin conexión[1]. Sin embargo, el proceso puede extenderse debido a un error que ocurre en ocasiones a la hora de construir el archivo de actualización. Una vez generados los archivos de actualización, éstos son copiados al servidor de producción donde el asistente de instalación en el computador de cada cliente los puede descargar y posteriormente combinar con la versión local de la colección que cada uno de ellos posee.

Todo el proceso es llevado a cabo en el servidor de trabajo. Este servidor contiene la colección sin procesar del cliente, almacenada en una estructura de directorios particular (es un reflejo de la tabla de contenidos). Adicionalmente, posee el ambiente de trabajo requerido el cuál debe coincidir con las rutas programadas en los scripts de Perl.

2.2 Frecuencia

El proceso empieza por decisión del cliente. El administrador de contenidos debe colocar un archivo directamente en el directorio *root* (con el nombre de *run.log*), lo cual hará que una tarea programada corriendo en el servidor de trabajo, cuya única tarea es revisar cada hora si un archivo nombrado *run.log* existe en el directorio *root*, inicie el proceso.

[1] Existen dos colecciones. Una para el contenido en línea y otra para distribución sin conexión. Sin embargo, ambas contienen exactamente el mismo contenido, lo que varía es la forma en que el cliente accede a él.

2.3 Volumen de Datos

En la colección hay alrededor de 30000 archivos. Éstos ocupan aproximadamente 1500 MB de espacio en disco. Una vez construida la colección de NXT, ésta ocupa alrededor de un 20% más de espacio en disco debido a los índices generados. El proceso de actualización tarda entre 45 minutos y 1 hora dependiendo de la cantidad archivos procesados.

2.4 Flujo General

La siguiente figura muestra de manera general, la secuencia de tareas, salidas y entradas para el proceso de actualización de contenidos para AILA.

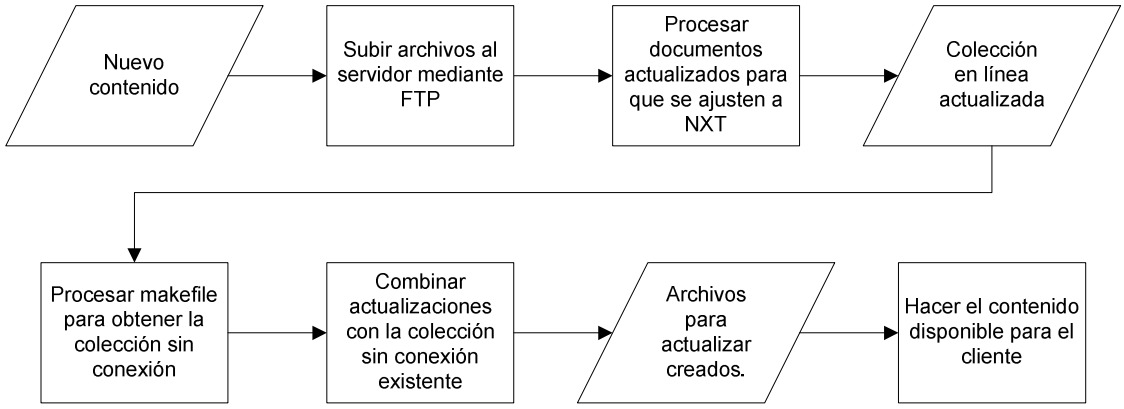


Figura 2.1 Flujo general del proceso[1].

[1] Ver apéndices para obtener la legenda y significado de los símbolos presentes en el diagrama de flujo.

2.5 Tareas

La siguiente sección muestra las tareas requeridas para completar la actualización de contenidos en línea y sin conexión respectivamente.

2.5.1 Colección en Línea

La figura siguiente muestra la secuencia de tareas que se requieren para completar la actualización del contenido en línea.

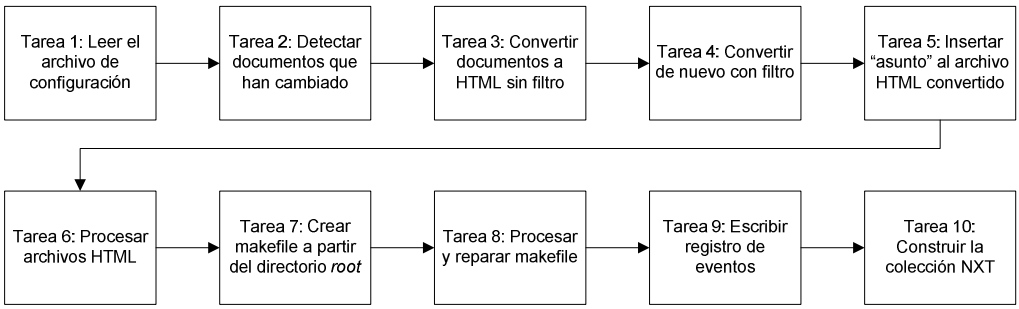


Figura 2.2 Flujo del proceso de actualización de contenidos en línea.

Como se puede observar en la Figura 2.2 el proceso se puede subdividir en diez tareas principales, que se podrían considerar como etapas del proceso. De estas diez, las tareas 1, 5,6 y 8 requieren del uso de expresiones regulares. Las tareas 3 y 4 utilizan un módulo escrito en Perl para hablar directamente con Microsoft Word 2003. La tarea 10 utiliza la herramienta ccBuild para construir la colección NXT a partir del makefile generado en las tareas 7 y 8. La tarea 2 determina si el documento de Microsoft Word 2003 leído, debe de convertirse a HTML o no. Para esto verifica si existe un archivo HTML con el mismo nombre, si la fecha de modificación de éste es menos reciente que la del documento de MS Word 2003 correspondiente entonces realiza la conversión de formato.

2.5.2 Colección sin Conexión

La figura siguiente muestra la secuencia de tareas que se requieren para completar la actualización del contenido sin conexión.

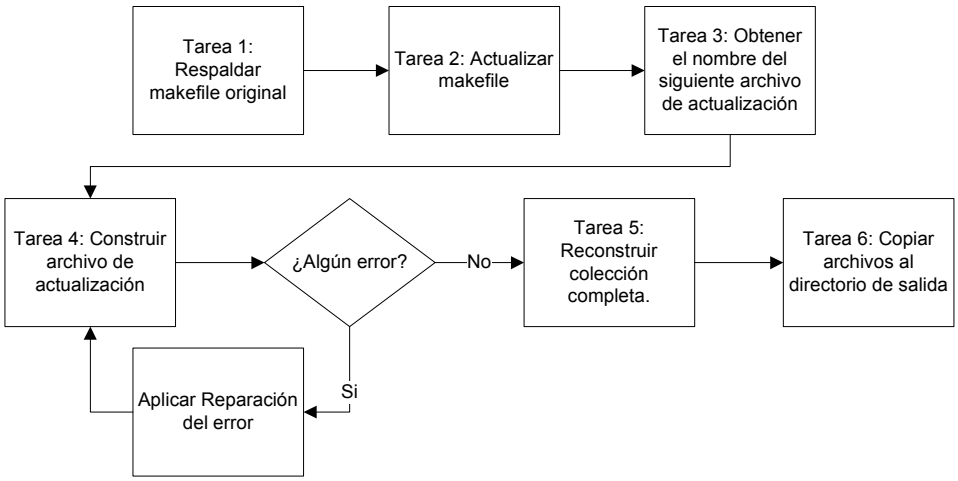


Figura 2.3 Flujo del proceso de actualización de contenidos sin conexión.

Según como se muestra en la Figura 2.3 el proceso de actualización de la colección sin conexión utiliza como entrada el makefile generado en el proceso descrito en la Figura 2.2. La tarea 2 utiliza expresiones regulares para actualizar las propiedades de la colección (cambia el título, ubicación del archivo NXT generado y agrega una contraseña). Las tareas 3 y 4 se encargan de construir el archivo de actualización siguiente. La tarea 5 invoca la herramienta Library Manager desde línea de comandos para reconstruir la biblioteca completa. Finalmente, la tarea 6 sencillamente copia los archivos de actualización generados al directorio de salida del proceso utilizando un archivo batch de Windows.

Nótese la existencia de una tarea llamada “Aplicar Reparación del Error”. Esta tarea deberá realizarse manualmente cada vez que ocurra un error a la hora de construir el siguiente archivo de actualización. Esta tarea se detalla en la siguiente figura:

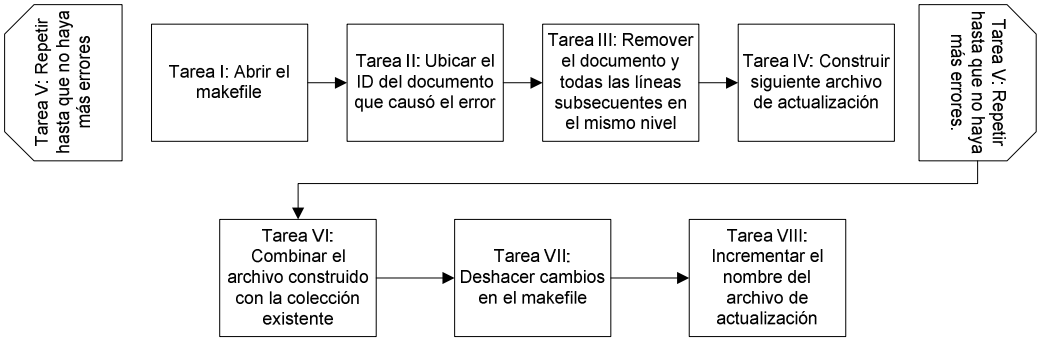


Figura 2.4 Flujo del proceso para reparar el archive de actualización

La reparación consiste en crear un archivo de actualización adicional, que eliminará de la colección existente cualquier situación que pueda causar el error nuevamente. Posteriormente, el archivo de actualización que se creará contiene reajustes a la tabla de contenidos (dado que documentos fueron borrados en la primera actualización) y el contenido actualizado. Como resultado, se obtienen dos archivos de actualización.

Capítulo 3
Diseño de la Solución

Esta sección describe el diseño de la alternativa realizada en Java. Se presenta una breve introducción a la plataforma Aspire y luego se presenta el diseño en la misma para el proceso de actualización de contenidos. Finalmente se muestra el diseño arquitectónico de los componentes implementados para Aspire.

3.1 Introducción a Aspire

Aspire es una plataforma de alto rendimiento que puede buscar documentos de diferentes fuentes, extraer meta datos y texto de dichos documentos, modificarlos y publicar la información recolectada a lo largo del proceso a motores de búsqueda, bases de datos o herramientas para publicación de contenido.

Los documentos encontrados son empaquetados en una tarea de procesamiento y guardados como un documento de Aspire, para posteriormente enviarlos a un pipeline de trabajo (o varios), el cual realiza las tareas necesarias sobre el documento. Algunas de las ventajas que Aspire provee para el procesamiento de documentos son:

a. Sincronización de tareas de procesamiento de documentos automática.
b. Manejo y coordinación de tareas de procesamiento de documentos.
c. Habilidad para dividir un documento y crear sub-tareas para cada elemento que sea encontrado.
d. Configuración dinámica de cambios.
e. Es posible añadir nuevos componentes dinámicamente.
f. Actualización del código de componentes dinámica.
g. Muchos métodos para el procesamiento de XML integrados.
h. Configuración jerárquica de componentes.
i. Administración utilizando una interfaz web fácil de comprender y con variadas opciones.

Los beneficios de utilizar Aspire para el proceso de actualización de contenidos para AILA se pueden resumir en la siguiente lista:

a. Únicamente los documentos que han cambiado serán procesados.
b. Es posible cargar dinámicamente componentes y su configuración, proporcionando un servicio ininterrumpido.
c. Proporciona mayor control del proceso. Éste iniciará en cuanto el cliente lo desee (no deberá esperar una hora como en el proceso actual).
d. Permite ver estadísticas más detalladas del proceso.
e. Se puede ver un registro detallado de errores y eventos del proceso.
f. Implementaciones para futuros clientes se realizarán en menor tiempo, debido a la amplia reutilización de componentes.
g. Tiempos de respuesta más rápidos (estadísticas y eventos pueden verse casi en tiempo real).

3.2 Descripción General

El proceso de actualización implementado en Aspire está compuesto por dos pipelines, uno para cada tipo de actualización (en línea/sin conexión). Un feeder[1] diferente fue configurado para cada pipeline de tal forma que ambos procesos puedan ser iniciados independientemente, sin embargo el feeder para el proceso de actualización sin conexión utilizará como fuente de documentos la salida del proceso de actualización en línea. Esto significa que el feeder para actualización en línea debe ser ejecutado primero si se quisiera realizar la actualización de contenidos sin conexión.

[1] Componente especial que permite obtener documentos de alguna fuente en particular. Véase la sección 3.4para obtener más información en los feeders usados en el proyecto.

El pipeline del proceso para actualización en línea encapsula gran parte del procedimiento requerido para los documentos de entrada. Esto incluye la conversión de formato (de documentos MS Word 2003 a HTML), la modificación de los documentos HTML convertidos para hacer que se ajusten al sitio NXT, la creación del makefile y la construcción de la colección NXT. Finalmente la colección construida es copiada al servidor de la herramienta de publicación, con el fin de que las actualizaciones sean visibles al cliente.

El proceso de actualización sin conexión usará entonces el makefile generado por el proceso previo y construirá archivos individuales de actualización que indican qué cambios realizar a la colección sin conexión existente. Si es necesario dicho proceso aplicará la reparación del error que ocurre debido a la convención de nombres utilizada con AILA.

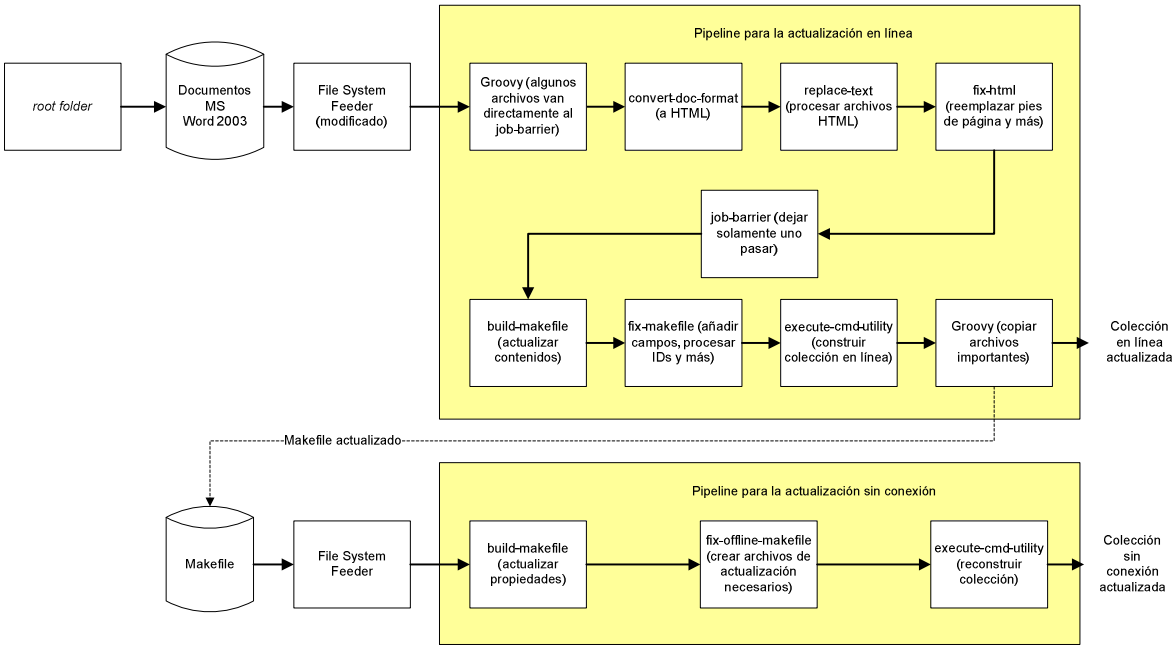


Figura 3.1 Pipelines utilizados para la implementación del proceso en Aspire.

Como se muestra en la figura anterior, el pipeline para la actualización de la colección en línea es alimentado por una variación del File System Feeder. Este feeder leerá todos los documentos Microsoft Word 2003 en el directorio *root* que han cambiado desde la última vez que se realizó una búsqueda y los etiquetará en el mismo lote de procesamiento. Conforme los documentos son procesados (aunque algunos no son procesados del todo), el stage job-barrier va reteniendo todos los documentos hasta que el último del mismo lote haya sido procesado. Únicamente el último documento en llegar a dicho stage continuará el resto del pipeline, y será éste el encargado de construir el makefile, invocar el comando ccBuild y copiar los archivos importantes (la colección NXT es copiada al servidor de publicación y el makefile es copiado al directorio de búsqueda configurado para el feeder del proceso sin conexión).

El pipeline para el proceso de actualización sin conexión, espera únicamente una tarea de procesamiento conteniendo el makefile actualizado (resultante del proceso de actualización en línea). La primera responsabilidad del pipeline será actualizar propiedades en el makefile recibido (nombre, ubicación y más), luego construir los archivos de actualización requeridos (a lo sumo dos) y finalmente volver a construir la colección sin conexión para combinar los nuevos cambios y copiar los archivos de actualización al directorio de salida. Dado que la herramienta para construir la colección desde línea de comandos presente en Library Manager copia los archivos de actualización por sí mismo, no son necesarios más stages.

Ambos pipelines pertenecen al mismo administrador de pipelines[1]. Dado que el proceso de actualización de la colección sin conexión no se ejecuta al mismo tiempo que el de la colección en línea, no es necesario crear un administrador de pipelines adicional para él. Crear un pipeline adicional podría incurrir en gastos innecesarios dado el costo de mantener un Thread Pool adicional[2].

[1] Componente de mayor alto nivel que un pipeline de procesamiento. Se encarga de sincronizar las tareas de procesamiento a lo largo de los distintos pipelines mediante una cola de espera y un Thread Pool.
[2] Para ver obtener más información respecto al uso del objeto Thread Pool en Java ver la siguiente página web: http://download.oracle.com/javase/tutorial/essential/concurrency/pools.html.

3.3 Pipelines

3.3.1 Pipeline para la Actualización de Contenidos en Línea

Este pipeline realiza casi todo el procesamiento requerido para los documentos. El primer paso del pipeline consiste en un script Groovy para revisar si el documento debería omitir ciertos stages. Si el documento introducido dentro del pipeline fue marcado como un borrado ("deleted") entonces éste irá directamente al stage *job-barrier*, ignorando cualquier otro tipo de procesamiento. Todos los demás documentos recorrerán el pipeline en su totalidad.

El segundo stage es de tipo *convert-doc-format* que será configurado para convertir documentos MS Word 2003 a HTML. Una vez se ha convertido el documento, el stage *replace-text* utilizará una serie de reglas de reemplazo para ajustar los archivos HTML al publicador de NXT. Después, el siguiente stage *fix-html* realizará arreglos adicionales necesarios para la correcta visualización de los archivos HTML en la herramienta de publicación.

Una vez que los documentos alcanzan el stage *job-barrier* serán marcados como completos, a excepción del último documento perteneciente al mismo lote. Esta barrera permitirá únicamente a la última tarea de procesamiento continuar con el resto del pipeline. Ésta última tarea se encargará de construir el makefile en el stage *build-makefile*, generando un nuevo makefile a partir del directorio *root* con el fin de que cualquier cambio realizado en la colección se vea reflejado en el makefile.

Cuando se construye el makefile, la estructura del directorio *root* será utilizada como tabla de contenidos. Así por ejemplo, las carpetas que estén directamente en el directorio *root* serán los títulos de más alto nivel en la tabla de contenidos, las carpetas y archivos dentro de éstas serán el segundo nivel en la tabla de contenidos y así sucesivamente.

El siguiente stage, del tipo *fix-makefile* se encargará de realizar cambios en el makefile específicos para AILA. Una vez el makefile está listo, se invocará la herramienta ccBuild utilizando el stage *execute-cmd-utility* para construir la colección NXT. Finalmente, un script Groovy se encargará de copiar el makefile al directorio de búsqueda del feeder sin conexión y la colección NXT recién construida al servidor de publicación.

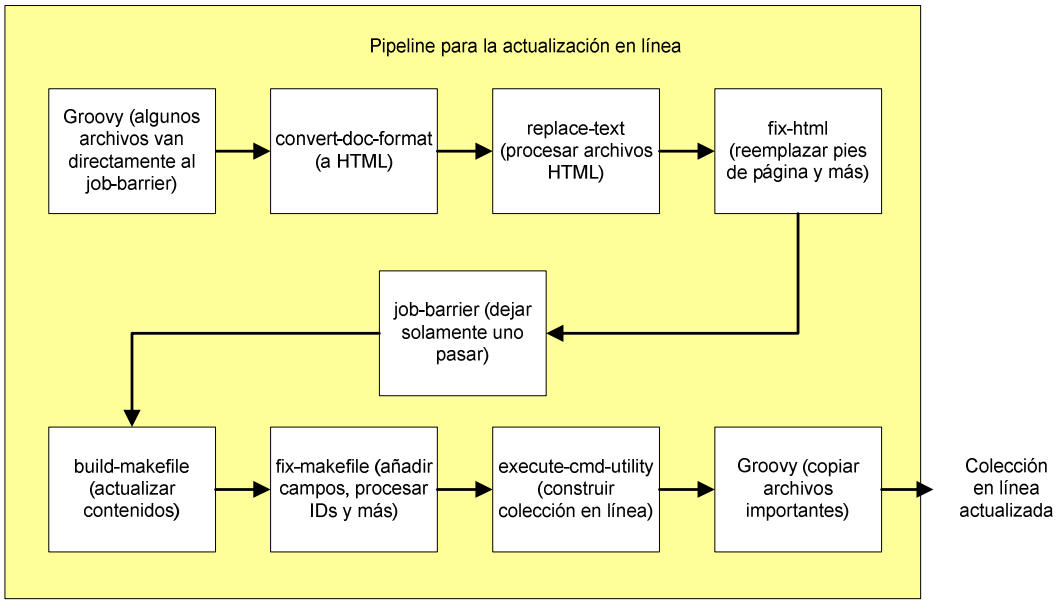


Figura 3.2 Pipeline para la actualización de contenido en línea.

3.3.2 Pipeline para la Actualización de Contenidos sin Conexión

Al igual que el pipeline anterior, éste será controlado directamente por el cliente. Esto significa que el administrador de contenido de AILA puede iniciar el proceso a voluntad y ver reporte de eventos y errores. Una vez el makefile ha sido generado por el pipeline anterior, éste será usado como entrada para el pipeline de actualización sin conexión (nótese que el cliente debe iniciar éste proceso y siempre deberá ser después de haber finalizado el pipeline en línea, aunque iniciarlo antes no tendrá ningún efecto).

El feeder del sistema de archivos buscará el makefile e informará si es nuevo. Si lo es, entonces un nuevo documento de Aspire es creado e introducido en el pipeline. El primer stage se encargará de actualizar propiedades en el pipeline para modificar el nombre, la ubicación y añadir una contraseña a la colección. Luego, un stage propio para AILA se encargará de construir los archivos de actualización requeridos, corrigiendo el error presente si es necesario. Finalmente, el *execute-cmd-utility* se encargará de reconstruir la colección utilizando Library Manager.

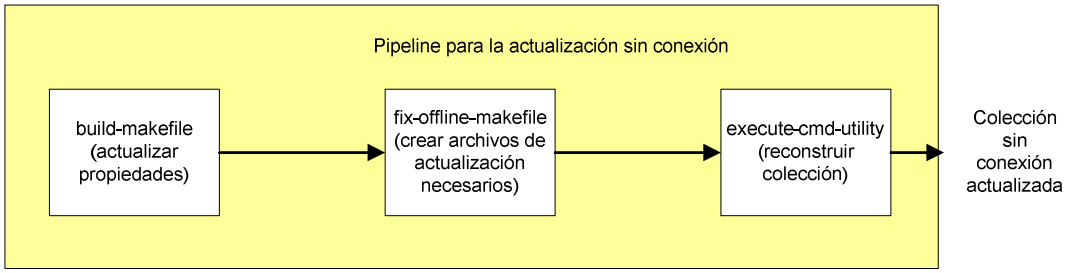


Figura 3.3 Pipeline para la actualización de contenido sin conexión.

3.4 Feeders

La siguiente sección describe los feeders utilizados en el proyecto. Es importante resaltar, que de estos únicamente la variación del File System Feeder fue implementada con el fin de ser usada con AILA, el componente original ya existía en el repositorio de componentes de Aspire.

Tabla 3.1 Descripción del File System Feeder

FILE SYSTEM FEEDER	
Entrada	Archivos en los directorios monitoreados.
Salida	Un documento de Aspire conteniendo la ruta al archivo descubierto en el directorio de búsqueda mapeada a las etiquetas <url> y <fetchUrl> y la acción del documento introducida como un atributo del mismo. Este documento es enviado al administrador de pipelines configurado.
Descripción	Periódicamente monitorea un número de directorios (y opcionalmente sub directorios) buscando archivos que hayan cambiado desde la última búsqueda. Cambios incluyen adiciones nuevas, modificaciones o borrados. Los archivos cambiados son ingresados en el administrador de pipelines de Aspire. El File System Feeder monitorea uno o más directorios, revisando si hay archivos actualizados (opcional incluir un filtro de archivos). El feeder construye una fotografía de la estructura del directorio (opcionalmente incluyendo sub directorios) y compara ésta contra la fotografía creada la vez anterior que el feeder revisó el directorio. De aquí en adelante, una lista de archivos nuevos, modificados y eliminados es construida, y esos archivos son enviados al administrador de pipelines de Aspire. Cuando todos los cambios de los directorios buscados han sido procesados, el feeder continúa con el directorio siguiente cuando no hay más directorios, el feeder dormirá por un período de tiempo antes de volver a revisar la misma lista de directorios de búsqueda.

Tabla 3.2 Descripción de la variación del File System Feeder utilizada en el proyecto.

FILE SYSTEM FEEDER (VARIACIÓN)	
Entrada	Archivos en los directorios de búsqueda.
Salida	Un documento de Aspire conteniendo la ruta al archivo encontrado en el o los directorios de búsqueda mapeada a las etiquetas <url> y <fetchUrl> junto con la acción correspondiente. Éste documento es enviado al administrador de pipelines para que lo introduzca en el pipeline indicado.
Descripción	Variación del File System Feeder que entregará al administrador de pipelines todos los documentos encontrados en los directorios de búsqueda al mismo tiempo y con el mismo identificador de lote. Posponer la entrega de documentos al administrador de pipelines hasta tenerlos todos, permite al feeder dar a conocer la cantidad de documentos que hay en un lote en particular, esto es necesario para que el stage *job-barrier* funcione correctamente.

3.5 Stages

A continuación se describe la funcionalidad de cada stage utilizado en el proyecto. Se separan aquellos que fueron implementados y aquellos que fueron reutilizados.

3.5.1 Stages Nuevos

Tabla 3.3 Descripción del stage *build-makefile*.

BUILD MAKEFILE	
Entradas	Un makefile existente para actualizar sus propiedades o la ruta a un directorio para utilizarlo como fuente de contenidos para el makefile.
Salidas	Makefile nuevo o actualizado escrito en el sistema de archivos.
Descripción	Lee la estructura de un directorio y crea un makefile a partir de ella. Si el makefile no existe, uno nuevo es creado; si un makefile existente es provisto, entonces será actualizado. Una plantilla de makefile debe especificarse para poder crear el makefile basado en ella (es obligatoria).

Tabla 3.4 Descripción del stage *convert-document-format.*

CONVERT DOCUMENT FORMAT	
Entradas	Documento de Aspire y un formato de conversión destino.
Salidas	Archivo convertido creado y almacenado en el sistema de archivos, la etiqueta <url> actualizada y meta datos extraídos agregados al documento de Aspire.
Descripción	Busca el documento señalado por la etiqueta <url> en el documento de Aspire y lo convierte al formato destino. Un nuevo archivo con el formato destino es creado, por ende conservando el archivo original. Extrae meta datos del archivo original y los mapea al documento de Aspire (depende de cada tipo de los formatos siendo convertidos y el motor de conversión que se esté usando). También un parámetro opcional que especifica en que carpeta colocar el archivo convertido puede ser provisto al stage. Este stage seleccionará dinámicamente el motor de conversión que se ajuste a los formatos de fuente y destino.

Tabla 3.5 Descripción del stage *execute-commandline-utility*.

EXECUTE COMMAND LINE UTILITY	
Entradas	Nombre del comando y sus parámetros. Los parámetros pueden ser una referencia a un elemento en el documento de Aspire.
Salidas	Salidas del comando ejecutado si existe algún error.
Descripción	Este stage ejecuta una utilidad de línea de comandos utilizando los parámetros provistos. Datos de entrada o parámetros son colocados en la configuración del stage y pueden ser referencias a un elemento del documento de Aspire. Este stage es independiente de la plataforma, sin embargo las utilidades que son invocadas desde él no. Esto significa que el comando especificado en la configuración del stage depende de la máquina que está ejecutando la aplicación de Aspire.

Tabla 3.6 Descripción del stage *replace-text*.

REPLACE TEXT	
Entradas	Documento de Aspire y una lista de reglas de reemplazo.
Salidas	Archivo referenciado por el documento de Aspire modificado con las reglas de reemplazo aplicadas sobre él.
Descripción	Aplica operaciones de búsqueda y reemplazo (reglas de reemplazo) a un documento. El texto buscado y la cadena substitución pueden ser patrones (expresiones regulares). La lista de reglas de reemplazo debe ser especificada en la configuración del stage. Los reemplazos pueden contener variables, cuyo contenido es almacenado en el documento de Aspire.

Tabla 3.7 Descripción del stage *job-barrier*.

JOB BARRIER	
Entradas	Una tarea de procesamiento de documentos.
Salidas	Depende de la configuración del stage. Puede dejar únicamente a una tarea de procesamiento continuar con el resto del pipeline o a todas. Siempre actualiza la información de las tareas de procesamiento que pasan a través de él con la lista de todas las tareas que llegaron hasta este stage. Es decir, cualquier tarea que haya continuado en el pipeline después de un stage *job-barrier* contendrá la lista de todas las otras tareas que llegaron a dicho stage (hayan continuado o no).

Descripción	Este stage hará tareas de procesamiento esperar hasta que todas las otras tareas pertenecientes al mismo lote lleguen al stage. Puede ser configurado para dejar a todas las tareas continuar o finalizar a todas excepto por la última. El objetivo de la barrera es permitir mayor sincronización de tareas (por ejemplo, cuando algunos documentos dependen de que otros hayan sido procesados para poder continuar). Para poder utilizar este stage, el feeder debe informar al administrador de pipelines que contiene el *job-barrier* la cantidad total de tareas de procesamiento que se ingresarán. Esto es requerido porque el manejador de barreras va a contar las tareas que alcancen este stage y basados en ese contador decidir si la tarea es la última en llegar o no.

Tabla 3.8 Descripción del stage *fix-html*.

FIX HTML	
Entradas	Ruta de un HTML existente en el sistema de archivos.
Salidas	Archivo HTML con contenidos actualizados.
Descripción	Este stage hará arreglos particulares a los archivos HTML que son requeridos para el proceso de actualización de contenidos de AILA únicamente. Estas reparaciones tienen como fin hacer que el HTML se ajuste al publicador de NXT. Hay dos arreglos mayores que son hechos, uno es la inserción de IDs que NXT puede comprender y el otro es el reemplazo de pies de página por pop-ups.

Tabla 3.9 Descripción del stage *fix-makefile*.

FIX MAKEFILE	
Entradas	Ruta a un makefile en el sistema de archivos.
Salidas	Makefile referenciado por la etiqueta <url> en el document de Aspire actualizado.
Descripción	Este stage realizará arreglos particulares al makefile, requeridos en el proceso de actualización de contenido de AILA. El trabajo hecho al makefile consiste en asignar IDs únicos a ciertos documentos y remover entradas innecesarias. Estas reparaciones son hechas con el fin de permitir que el contenido publicado en NXT tenga el formato correcto.

Tabla 3.10 Descripción del stage *fix-offline-makefile*.

FIX OFFLINE MAKEFILE	
Entradas	Ruta de un makefile en el sistema de archivos.
Salidas	Uno o dos archivos de actualización secuenciales siguientes (el makefile no se modifica).
Descripción	Utiliza el makefile de la actualización sin conexión y aplica la corrección del error hasta que no haya más errores. Esta corrección consiste en tratar de construir el siguiente archivo de actualización, si falla, entonces elimina cualquier documento que haya dado problemas. Entonces, vuelve a crear el archivo de actualización. Una vez creado, lo combinará con la colección existente y luego creará otro archivo de actualización que corresponde al siguiente en la secuencia. Así, este stage creará un archivo de actualización si no se encontraron errores y dos en caso de que sí.

3.6 Diagramas de Clase

La siguiente sección muestra los diagramas de clase para los componentes que fueron desarrollados en el proyecto. Los siguientes diagramas mostrarán la descripción de nuevas clases, atributos y métodos que fueron agregados o modificados a la arquitectura existente de Aspire.

3.6.1 File Feeder (modificación)

Este feeder hereda de las clases SimpleFeederImpl y FileFeederImpl. SimpleFeederImpl fue modificado al agregar un nuevo método que establece la cantidad total de tareas de procesamiento que el feeder va a ingresar como un mismo lote. Note también que un nuevo método es añadido a lo lago de la cadena de invocaciones hasta llegar a la clase PipelineManagerImpl con el fin de soportar la sincronización tipo barrera.

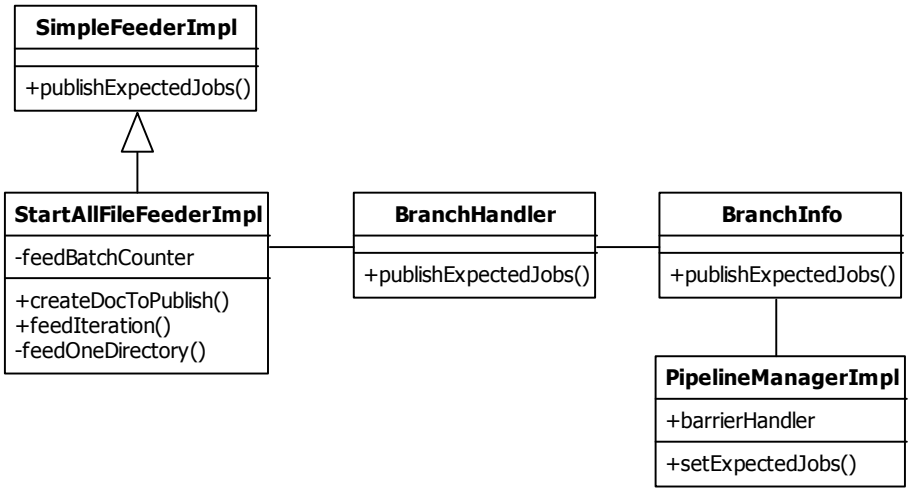


Figura 3.4 Diagrama de clases de la variación del File System Feeder

3.6.2 Convert Document Format

El stage convertidor recibirá un documento y un formato destino, utilizando la clase ConverterFactory determinará que implementación de Converter concreta utilizar para poder efectuar la conversión necesaria.

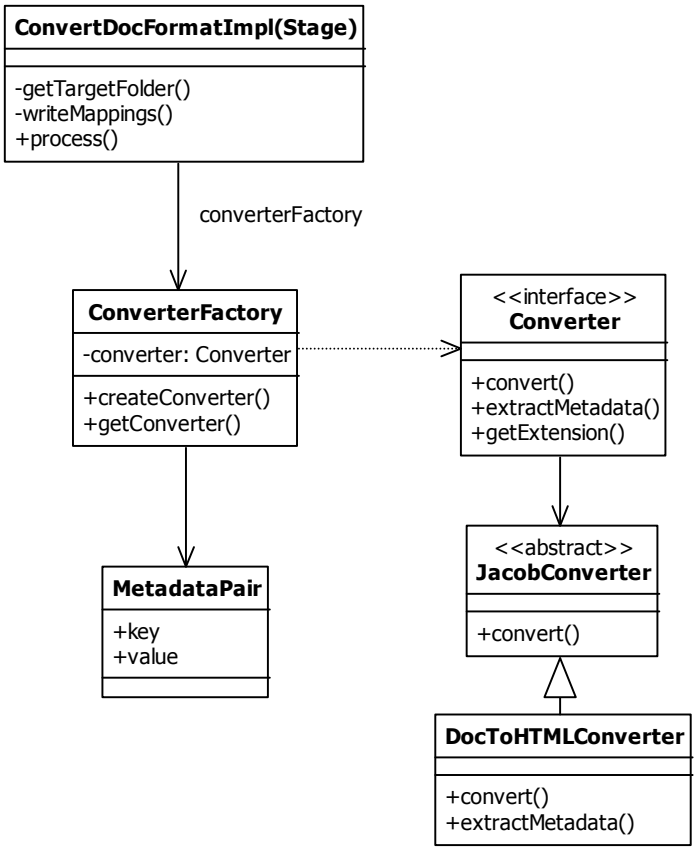


Figura 3.5 Diagrama de clases del stage *convert-document-format*.

3.6.3 Replace Text

Este stage requiere la definición de un conjunto de reglas de reemplazo que serán aplicadas a cada documento recibido. Esta lista de expresiones y reemplazos es cargada en la inicialización del stage.

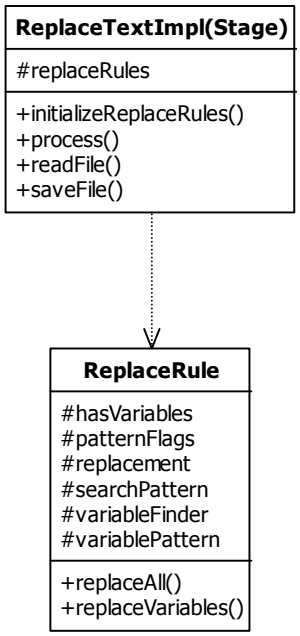


Figura 3.6 Diagrama de clases del stage *replace-text.*

3.6.4 Job Barrier

Cuando una tarea de procesamiento[1] alcanza este stage, ésta será pausada hasta que todas las demás tareas introducidas por el feeder en el mismo lote lleguen al stage. El funcionamiento de este stage es similar al del patrón de barrera utilizado para sincronización de hilos, pero aplicada a un nivel de tarea de procesamiento (en realidad nunca se modifica el comportamiento de los hilos).

[1] Tarea de procesamiento hace referencia a la clase Job en la Figura 3.7 Diagrama de clases del stage *job-barrier*.Figura 3.7.

Una nueva clase, BarrierHandler es introducida con el fin de administrar la sincronización e información de la barrera. Una instancia de esta clase es creada por la clase PipelineManagerImpl utilizando el nuevo método initBarrier(). Una vez el manejador de barreras es inicializado, cuando una tarea de procesamiento invoque el método de await, la bandera isPaused de la tarea de procesamiento es cambiada a “verdadero”; entonces la tarea es agregada a la lista arrivedJobList y su ejecución terminada, liberando el hilo que a cargo de dicha tarea para que otra tarea distinta pueda ser traída hasta la barrera. Cuando la última tarea de procesamiento ejecuta el método await(), el administrador de barreras añadirá todas las tareas en espera (o la última) a la cola del administrador de pipelines, permitiendo la ejecución del resto del pipeline por parte de esas tareas en espera.

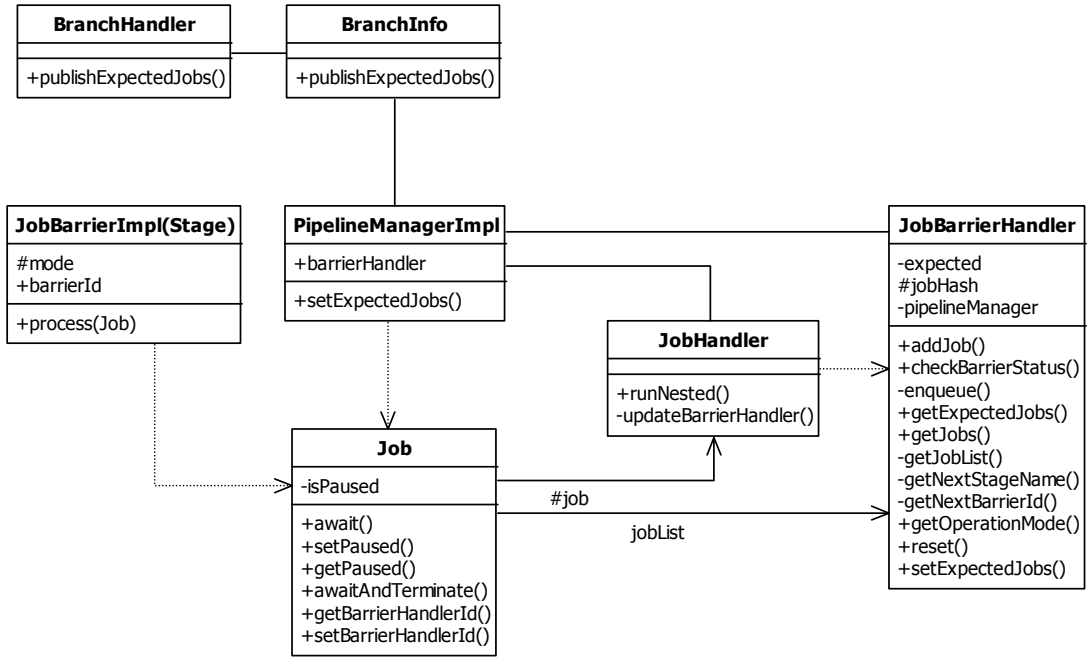


Figura 3.7 Diagrama de clases del stage *job-barrier*.

3.6.5 Build Makefile

Este stage hace uso de funciones integradas para el procesamiento de archivos XML para construir el makefile o actualizarlo.

Figura 3.8 Diagrama de clases del stage *build-makefile*.

3.6.6 Execute Command Line Utility

Este stage utiliza una nueva clase que permite ejecutar un comando y capturar sus salidas para determinar si dicha ejecución fue exitosa o no.

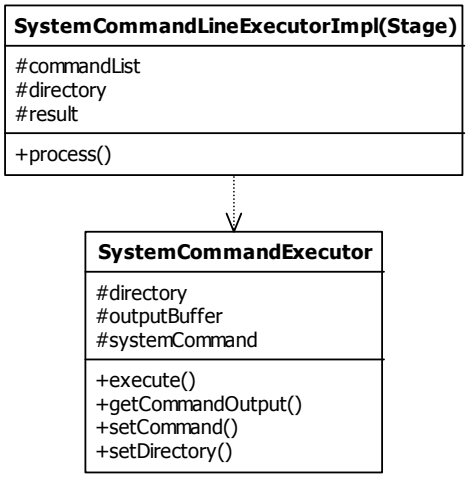


Figura 3.9 Diagrama de clases del stage *execute-commandline-utility*.

3.7 Interfaces de Usuario

La única interfaz necesaria entre el cliente y el servidor de Aspire (donde los pipelines implementados estarán ejecutando) es para iniciar o detener el proceso y ver eventos. Toda la información requerida por el cliente y las opciones de control están en una sola página web, parte del sitio de administración del cliente existente.

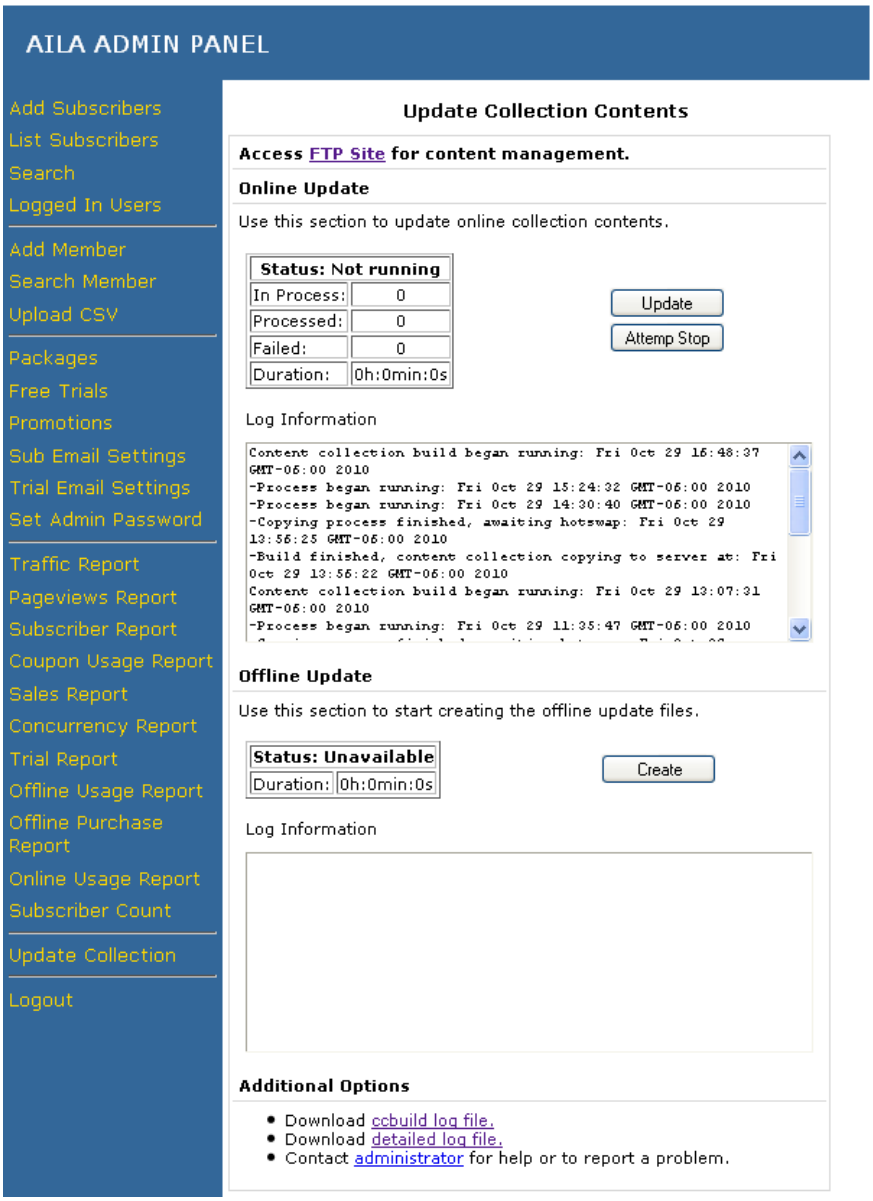


Figura 3.10 Página para la actualización de contenidos del cliente

La Figura 3.10 muestra una captura de pantalla de la página web utilizada para comunicar el cliente con el servidor de Aspire. Esta página contiene información del proceso de actualización, registro de eventos y opciones de control. La siguiente tabla explica el significado de cada campo en la tabla de estado y los posibles valores de la misma:

Tabla 3.11 Leyenda de la tabla de estado

CAMPO	SIGNIFICADO	VALORES POSIBLES
Status	Indica el estado actual del proceso de actualización.	Puede ser uno de: “Running”, “Not Running”, “Stopping” y “Unavailable”. El último es en caso de que la conexión al servidor de aspire no pueda ser establecida.
In Process	Muestra cuantos documentos a la vez están siendo procesados actualmente por el servidor de Aspire.	Número entero no negativo. Este número representa normalmente la sume de tareas en la cola del administrador de pipelines más la cantidad total de hilos configurada.
Processed	Contador de cuantos documentos ya han sido procesados.	Entero positivo.
Failed	Indica cuantos documentos no se pudieron procesar debido a alguna falla hasta el momento.	Entero mayor o igual que cero.
Duration	Muestra el total de tiempo transcurrido desde el inicio del proceso.	Horas, minutos y segundos.

Nótese que todos los valores descritos en Tabla 3.11 son limpiados cada vez que el usuario hace clic en el botón de Update o Create. Estos botones empiezan el proceso de actualización en línea y sin conexión respectivamente invocando el comando de inicio en el Servlet registrador por el feeder correspondiente en el servidor de Aspire.

El botón de Stop invocará el comando de parar en el Servlet del feeder en línea. Esto causará que el feeder se detenga inmediatamente, sin embargo otros documentos que ya fueron introducidos en el pipeline no serán detenidos. Finalmente, las cajas de información de registro muestran eventos conforme van ocurriendo en el proceso.

3.8 Servicios y Componentes

Esta sección provee una explicación de los componentes y servicios que fueron implementados o usados con el propósito de hacer que la aplicación de Aspire funcione correctamente.

La página web del cliente se comunica con el servicio de OSGI HTTP por medio de comandos HTTP (Servlet). Estos comandos retornan respuestas XML que son analizados y desplegados en la página web. Los servicios OSGI coordinan la creación y operación de ambos el pipeline de actualización en línea y sin conexión. Software de Microsoft Office es accedido por el pipeline de actualización en línea utilizando interfaces COM.

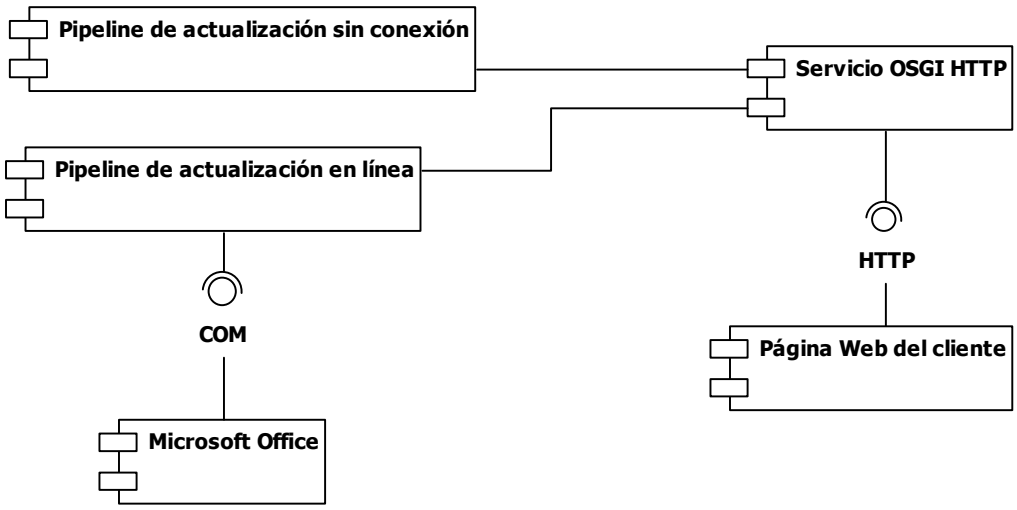


Figura 3.11 Diagrama de componentes y servicios

Capítulo 4
Análisis de Resultados

La siguiente sección realiza un análisis de los resultados obtenidos al migrar el proceso de actualización de contenido de Perl a Java. Se detallan aspectos más importantes que cambiaron la forma en que el proceso se realizaba en Perl, así como estadísticas de tiempo de ejecución. Finalmente se muestra muestran de las colecciones construidas con cada proceso para demostrar la eficacia del proceso implementado con Aspire.

4.1 Comparación entre Procesos

4.1.1 Automatización

El proceso actual escrito en Perl realiza la primera parte de la actualización (para la colección en línea) automáticamente, pero una vez que ésta ha sido completada el personal de Search Technologies debe hacer manualmente la actualización para la colección sin conexión.

Esta es quizás la diferencia más evidente entre ambos procesos, el proceso implementado en Aspire es completamente automático. Es decir, que la intervención humana no es necesaria desde el momento en que el cliente presiona el botón de inicio del proceso hasta que éste termina.

4.1.2Eficiencia

La siguiente tabla muestra el análisis de rendimiento realizado para ambos procesos. Este análisis toma en cuenta la construcción de la colección completa.

4.1 Comparación del tiempo de ejecución de ambos procesos[1].

Etapa	Tiempo en Perl (segundos)	Tiempo en Aspire (segundos)	Porcentaje de mejora potencial
Procesar archivos	6074.4	3068.4	197%
Construir colección	1684.8	1971.6	85.4%
Total	7759.2	5040	153.9%

La tabla anterior, resume los resultados obtenidos de ejecutar el proceso en Perl y en Aspire cinco veces, utilizando la colección entera como entrada. Es posible observar que el proceso ejecutado en Aspire puede aportar una mejora potencial de alrededor de 197% al procesamiento de documentos. Se atribuye esta mejora a la técnica de múltiples hilos ejecutando simultáneamente (y por ende convirtiendo concurrentemente documentos de Word) y a un manejo más eficiente de los archivos XML (cuando se construye el makefile).

Respecto a la construcción de la colección se observa una reducción del rendimiento de un 14.6%. Esta penalización es causada por un incremento en el tamaño del archivo makefile, ocasionada por la utilización de ciertos métodos para el análisis de XML pertenecientes a Java. Estos métodos leen el archivo DTD declarado por el makefile y crea todos los atributos especificados ahí con los valores por defecto. Cuando se escribe el makefile en disco (se utiliza DOM para procesa el makefile), éste incluye más atributos que el original, por eso su incremento en el tamaño.

En general y aunque la construcción de la colección se ve penalizada, es posible obtener una mejora del 53.9% en el tiempo de ejecución del proceso de actualización de contenidos.

[1] Ver apéndices para obtener detalles del ambiente en que se han realizado las comparaciones mostradas en ésta tabla.

Para el proceso de actualización de la colección sin conexión, no fue posible obtener una comparación objetiva del tiempo de ejecución de ambos procesos. Esto debido a que el proceso escrito en Perl es realizado manualmente y éste depende de la persona que esté llevando a cabo el proceso y la cantidad de errores que aparezcan en la construcción de los archivos de actualización.

4.2 Observaciones

Perl es un lenguaje de programación de alto nivel con una herencia diversa escrita por miles de contribuidores. Se deriva del lenguaje de programación C, de las herramientas sed, awk, el Shell de Unix y otros lenguajes y herramientas. Es ideal para prototipos rápidos, utilidades del sistema, herramientas de software, tareas manejadoras del sistema, acceso a bases de datos, programación gráfica, redes, y programación web[1].

Mientras tanto, Java es un lenguaje de programación de alto nivel multipropósito cuyas características principales se resumen en: simplicidad, neutro arquitectónicamente, orientado a objetos, portable, distribuido, alto rendimiento, multihilo, robusto, dinámico y seguro[2].

Mientras que Perl se origina como un lenguaje de scripting rápido y especializado, Java nace con el fin de permitir a los programadores desarrollar aplicaciones extensas y complejas. Sin embargo, la generalidad que proporciona un lenguaje como Java, tiene su costo en el rendimiento. A la hora de realizar la migración estas diferencias se hacen más evidentes, principalmente cuando se trata de migrar el procesamiento de texto (una de las fortalezas de Perl) a Java. Sin embargo esto no ha significado un problema serio en la migración, dado que se obtuvo ganancias en tiempo de ejecución en otras áreas del proceso. Por ejemplo, la construcción y reparación del makefile.

[1] Ver http://perldoc.perl.org/perlfaq1.html#What-is-Perl%3f para más información.
[2] Ver http://java.sun.com/docs/white/langenv/ para más información.

4.3 Conclusiones

a. Dependiendo del tipo de migración que se desea, es útil comprender el proceso que se desea migrar, para planificar la dicha migración y lograr dar un valor agregado al nuevo proceso. Este valor puede ser mejor rendimiento, ahorro de espacio en disco duro o bien mejor experiencia para el usuario.
b. Una migración directa de código escrito en Perl a Java puede resultar en pérdida de rendimiento. Principalmente cuando se migran operaciones en las cuales Perl se especializa tal y como expresiones regulares y procesamiento de archivos de texto.
c. En este proyecto fue posible conocer cómo funcionan los procesos para procesamiento de documentos. Se entendió la complejidad de los mismos y cómo se solucionan problemas típicos en este tipo de tareas (casos muy particulares que requieren atención adicional, cuellos de botella, reporte del proceso, entre otros).
d. Se obtiene un mayor entendimiento de cómo funcionan las expresiones regulares, sus implicaciones en rendimiento y cómo se pueden escribir programas que las utilicen de mejor manera. Además se obtiene mayor experiencia en programación concurrente.
e. Conocimientos adquiridos en el curso Recuperación de Información Textual fueron de gran ayuda a la hora de realizar este proyecto. En primera instancia, la experiencia adquirida en el lenguaje de programación Perl a lo largo del curso facilitó el entendimiento de los procesos actuales, escritos en dicho lenguaje. La noción existente de procesos para indexar colecciones y procesar consultas a un motor de búsqueda obtenidas en el curso, fueron de ayuda a la hora de entender las necesidades del cliente y la estructura de la plataforma Aspire. Finalmente, la práctica conseguida en el curso en procesamiento de cadenas de texto, estructuras como arreglos y tablas hash ayudaron en la programación de los nuevos stages utilizados en el proyecto.

4.4 Recomendaciones

a. Ir a través del proceso que se pretende migrar manualmente provee detalles del proceso que no son visibles de otra manera. Tal es el caso de errores o pequeños ajustes requeridos.
b. Tomar estadísticas de tiempo y espacio en disco duro del proceso que se pretende migrar, con el fin de comparar una vez realizada la migración si dicho proceso ha mejorado en estos aspectos.
c. A la hora de migrar código de Perl a Java, es posible que se de cierta pérdida en el rendimiento por la falta de especialización que Java posee. Es recomendable buscar heurísticas que permitan omitir cierto procesamiento o añadan mejoras en el código original de Perl para que una vez migrado a Java se contrarreste cualquier posible pérdida de rendimiento.
d. Si se va a diseñar una adición a un sistema existente (y si este es orientado a objetos), es mejor utilizar las clases existentes y expandirlas que crear nuevas clases para agregar la nueva funcionalidad. Esto permitirá hacer las clases existentes más robustas y minimizar el acoplamiento.

Apéndices

4.5 Diccionario de términos

- **Archivo de actualización**

  Archivo generado por ccBuild. Este archivo contiene un conjunto de cambios que deben realizarse a una colección NXT existente. Normalmente es más pequeño que la colección existente.

- **Aspire**

  Plataforma escrita en Java para el procesamiento de documentos en un ambiente de alto rendimiento.

- **Biblioteca NXT**

  Conjunto de colecciones NXT.

- **ccBuild**

  Es una utilidad invocada desde línea de comandos que habilita la construcción de colecciones de contenido a partir de archivos ubicados en el sistema de archivos. Produce una colección NXT usando un makefile como entrada, o bien archivos de actualización. Esta herramienta es incluida en el software NXT Builder.

- **Colección NXT**

  Archivo completamente indexado, transportable, auto contenido, discreto y seguro que contiene una colección de documentos en una estructura jerárquica. Todos los documentos que residen en la colección yacen en su formato nativo y pueden ser navegados y recuperados desde un sitio NXT.

- **Expresión Regular**

  En computación, es una forma concisa y flexible para determinar si una cadena de texto coincide con un patrón en particular. Útiles para hacer reemplazos textuales.

- **FTP**

  Protocolo para la transferencia de archivos. Es utilizado por el administrador de contenido del cliente para subir los documentos al servidor de procesamiento (al directorio root).

- **Library Manager**

  Herramienta gráfica que permite crear, editar o construir librarías de NXT para su posterior publicación.

- **Makefile**

  El makefile de la colección de contenido es una pieza clave que habilita a ccBuild para construir colecciones NXT. Un makefile es un documento XML que contiene instrucciones que ccBuild sigue para construir una colección. Existe una correspondencia uno a uno entre la colección y el makefile; lo que significa, que para cada colección hay un makefile específico y viceversa. Un makefile contiene información relevante y necesaria con respecto a la colección en general, y lista la información respectiva a cada documento que ccBuild incluirá en la colección. Esa información incluye: ubicación del contenido fuente, estructura de la tabla de contenidos, opciones de cómo se verán los documentos, manejo de contenido, identificación y otros.

Debido a que los conflictos son manejados dentro del makefile, hay un control total de casi cualquier aspecto referente al contenido. El makefile debe satisfacer las guías establecidas en el archivo DTD y debe ser codificado para UTF-8.

- **PDF**

  Formato de documento portable. Ideal para publicar contenido en línea.

- **Sitio NXT**

  Es un repositorio de contenido que es alojado en el servidor NXT. El sitio puede estar organizado jerárquicamente usando carpetas y estructuras de contenido. Un sitio podría contener colecciones; servicios de contenido; enlaces a redes de contenido; plantillas de búsqueda; plantillas de despliegue; y propiedades del sitio, carpetas y colecciones de contenido.

- **Rocket Folio**

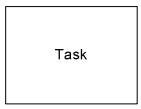
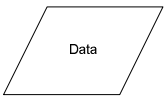
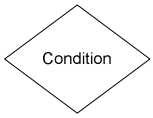
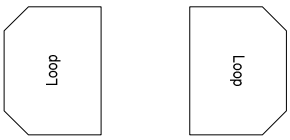
  Producto de Software que provee herramientas para generar infobases a partir de información y añade características de valor agregado que ayudan a hacer la infobase accesible a aquellos que la usan (publicador).

- **Rocket NXT**

  Suite para la publicación de contenido que da facilidades para compilar y distribuir información electrónica en línea o sin conexión.

4.6 Leyenda de Símbolos en el Diagrama de Flujo

El siguiente apéndice describe la notación usada en los diagramas de flujo.

Figura	Descripción
	Tarea o proceso. Representa algún trabajo que debe realizarse.
	Describe una fuente o salida de datos. En general significa que hay datos que son importantes en el proceso que son utilizados por alguna tarea posterior o que es el resultado de una previa.
	Condición. Usado para la toma de decisiones a lo largo del proceso.
	Delimitadores de un ciclo. Representan el inicio y fin de un bucle respectivamente.
	Indica la dirección del flujo.

4.7 Índice de Figuras

4.8 Índice de Tablas

4.9 Ambiente de Ejecución de las Pruebas

a. Sistema: Windows XP Professional Service Pack 3 32bits.
b. Procesador: Intel Core 2 T7200 2.00GHz.
c. 2.00 GB de memoria RAM.
d. Microsoft Office 2003.

4.10 Bibliografía

- Allen, J. perldoc.perl.org. *Perl Programming Documentation*. (Perl 5 versión 12.1). Recuperado de http://perldoc.perl.org/perl.html. (26-07-2010).
- Microsoft. Última Revisión: Agosto 5, 2004. Microsoft Support. *HOW TO: Use the HTML Filter in Word 2000*. (Revision 1.2). Recuperado de http://support.microsoft.com/?scid=kb;en-us;236967&x=11&y=6. (26-07-2010).
- FAST Search & Transfer ASA. *FAST NXT[4] Documentation*. Disponible al instalar Rocket NXT online server.
- Search Technologies.2010. *Aspire Javadoc*. (0.3-SNAPSHOT) Puede ser consultado en el sitio web https://wiki.searchtechnologies.com/javadoc/. Requiere credenciales.
- Search Technologies. Última revisión: 17 de Julio 2010. *Search Technologies Wiki*. Puede ser consultado en la página web https://wiki.searchtechnologies.com/mediawiki/index.php/Main_Page. Requiere credenciales.
- Oracle. *Java SE APIs & Documentation*. Recuperado de http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html (05-10-2010).