INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN

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

SOFÍA HO LIN

CARTAGO, NOVIEMBRE 2010

RESUMEN

El concepto de recuperación de información surgió en los años 50 por la sobrecarga informativa en el mundo informático. La necesidad de poder acceder a esta gran cantidad de información creó una demanda de sistemas automatizados de recuperación de información. Y aunque actualmente existen varias herramientas que permiten a los creadores de contenidos crear, manipular, buscar, consultar, y distribuir su información a otros, la búsqueda por información útil continúa siendo una ciencia que aún falta de exploración.

Search Technologies es una empresa que se especializa en proveer soluciones dentro de este ámbito. Como proyecto de práctica de especialidad se desarrolló un proceso de actualización para permitir a miembros de Aldrich Bonnefin & Moore, una firma concentrada en instituciones financieros y bancarios, mantener actualizados sus manuales de procedimientos estándares. Esto con el fin de que la información provista fuera exacta, oportuna, relevante y confiable.

El presente informe expone el plan de trabajo implementado para la migración de conversión del proceso en Perl a Aspire, el *framework* de la empresa. El proyecto se dividió en tres fases:

La primera fase consistió en el análisis del proceso existente basado en Perl y la documentación de la misma para identificar los requerimientos y entender el contexto del problema. La segunda fase se basó en el diseño. Se generó un documento con la propuesta de desarrollo que definía los módulos a implementar para reproducir este proceso de actualización en Aspire. Y por último, la tercera etapa consistió en el desarrollo e implementación de los módulos para la ejecución del proceso.

Palabras claves

Actualización; Aldrich Bonnefin & Moore; Búsqueda; Creadores de Contenido; Información; Migración; Módulos; Recuperación de Información; Search Technologies; Sistemas Automatizados.

ABSTRACT

The concept of information retrieval emerged in the 50s due to the overload in the world of informatics. The need to access this huge amount of information created demand for automated systems of information retrieval. Even though we can find several tools that allow content creators to create, manipulate, search, consult and distribute their information nowadays, the search for useful information is still a science that requires further exploration.

Search Technologies is a company that specializes in providing solutions in this area. As a specialty practice project, an update process was developed to allow members of Aldrich Bonnefin & Moore, a banking and business law firm, to keep their standard manual procedures updated so the information provided is exact, fortune, relevant and reliable.

The present document presents the project plan carried out to migrate this process from Perl to Aspire, the framework of the company. The project was divided in three phases:

The first phase consisted in the analysis of the existent process based in Perl and the documentation of the process to identify the needs and requirements of the client. The second part consisted in the design of Aspire modules that would be needed to reproduce the update process. And lastly, the third stage of this project consisted of development and implementation of the update process along with the respective documentation.

Keywords

Aldrich Bonnefin & Moore; Automated Systems; Content Creators; Information; Information Retrieval; Migration; Modules; Search; Search Technologies; Update.

Índice General

Capítulo 1

Introducción

1.1 Propósito

Este documento presenta una descripción detallada sobre la solución propuesta para la migración del proceso de actualización del cliente Aldirch Bonnefin & Moore en Perl al framework[1] de la empresa Search Technologies conocida como Aspire. Asimismo busca resaltar la existencia de tanto tecnologías que se utilizan y empresas que se enfocan en un área especializada para llevar a cabo la automatización de un proceso.

[1] Una arquitectura de software o conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular.

1.2 Contexto del Proyecto

Search Technologies es una empresa especializada en tecnologías de motores de búsqueda fundada en el año 2002. Desde sus inicios la empresa ha trabajado con las mejores herramientas que el mercado ofrece en este ámbito. Herramientas tales como FAST ESP, SharePoint y GSA (Google Search Appliance) eran utilizadas, pero aun así siempre se presentaban dos problemas: Primero, los costos muy elevados de la licencia del software dejaba la implementación del plan con bajo presupuesto. Y segundo, al utilizar un producto muy orientado a la tecnología, algunos objetivos del proyecto quedaban por fuera.

A raíz de esta problemática nació Aspire. Aspire, una plataforma desarrollada en Java, es un sistema de alto-rendimiento para procesar documentos y puede ser integrado con distintas herramientas Open Source como Solr, Lucene, Heritrix, y entre otras, para brindar una solución personalizada y de bajo costo a los clientes.

En diciembre del año 2009, Search Technologies adquirió una empresa llamada Info Solutions, ubicada en el estado de Kentucky, Estados Unidos. La labor de esta empresa era similar a la de Search Technologies sin embargo a una escala menor, utilizando tecnologías más sencillas y de bajo costo.

La mayoría de los clientes obtenidos por medio de esta adquisición, proveen documentos en formatos digitales planos tales como los de Microsoft Office Word o PDF para ir bajo una serie de procesos basados primordialmente en Perl para convertir estos documentos a un formato más estructurado que eventualmente es indexado por una tecnología de Rocket NXT[1] para facilitar la visualización y la búsqueda de información.

[1] Para más información, ver http://www.rocketsoftware.com/

Uno de los clientes, Aldrich Bonnefin & Moore[1], es una firma que se concentra en instituciones financieros y bancarios para proveer asistencia financiera corporativa, operacional, comercial, de consumidor y conforme a las regulaciones. Poseen una cantidad sustancial de manuales con procedimientos estándares en formato de Microsoft Office Word y PDF los cuales desean publicar para que su contenido pueda ser accedido por sus lectores. Sin embargo, estos manuales son constantemente actualizados con el fin de que la información provista sea la más exacta, actualizada y confiable. Y por lo tanto, Info Solutions creó un proceso de actualización.

El proceso desarrollado por la antigua Info Solutions consiste en la ejecución de algunas tareas manuales y de la ejecución de varios scripts de Perl[2] que llevan a cabo las transformaciones necesarias para poder publicar la información. Dado que es un script generado específicamente para este cliente, no se puede reutilizar sin tener que recurrir a editar el código fuente. Aunque este proceso se ha ido refinando y funciona correctamente, Search Technologies desea migrar este proceso a Aspire para mejorar el proceso tanto en rendimiento, reusabilidad y modularidad por medio de la automatización mediante Aspire.

Este proyecto de práctica consistió en migrar el proceso de actualización de Perl 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. Este proyecto fue realizado para el Departamento de Soporte de Search Technologies, sede de Costa Rica.

[1] Para más información, ver http://www.abmlawfirm.com/
[2] Un lenguaje de programación práctico para extraer información de archivos de texto y generar informes a partir de un contenido de los ficheros.

1.3 Descripción del Problema

Para actualizar el contenido de la colección de manuales de Aldrich Bonnefin & Moore, un encargado de soporte de Info Solutions crea un respaldo de los documentos existentes y copia los documentos nuevos dentro de la carpeta del manual correspondiente. Debe ubicar dentro de cada carpeta: la portada, el manual, los apéndices y la tabla de autoridades. Si el documento ya existe, se sobre-escribe.

Antes de ejecutar el script de Perl, debe revisar que los archivos que contienen los títulos de los manuales y apéndices, estén actualizados y cambiados de acorde a los archivos actualizados. Si estos están acorde a los archivos de la colección, debe ir a la carpeta sobre el cual trabaja el proceso y debe eliminar las carpetas de los manuales que van a ser actualizados. De esta forma, el ambiente queda listo para ejecutar el script de Perl, el cual efectúa una serie de procesamientos de texto para generar un archivo de colección.

Este proceso es tedioso, lo que conlleva a una mal utilización del tiempo y del recurso humano invertido en la actualización de la colección. Este problema afecta directamente al cliente, ya que depende del encargado de soporte para poder llevar a cabo una actualización en su información. También afecta al encargado de soporte dado que debe efectuar procesos manuales cuando estos pueden ser automatizados.

La solución propuesta es migrar el proceso de actualización existente a Aspire de tal manera que requiera únicamente de intervención humana para la supervisión del proceso y procesos de aseguramiento de calidad. Se espera que esta solución presente un ahorro significativo de tiempo que podrá ser invertido en otras actividades más productivas.

1.4 Involucrados

A continuación se muestra una tabla 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
Luis Pintor	Director de Soporte	Administrador del proyecto
Jonathan González	Desarrollador	Consultor en Aspire
Luis Luna	Desarrollador	Consultor en Aspire
Bob Berberich	Jefe de Operaciones en Kentucky	Soporte y Coordinación desde Kentucky
Tim Naylor	Desarrollador	Entrenador para NXT y Folio
Steve Miller	Desarrollador	Consultor en los procesos de Perl.
Stan Weldy	Desarrollador	Consultor en los procesos de Perl.
Tim Back	Desarrollador	Consultor en los procesos de Perl.

1.5 Objetivos

1.5.1 General

Migrar el proceso de actualización de contenido existente para Aldrich Bonnefin & Moore a la plataforma de procesamiento de documentos Aspire.

1.5.2 Específicos

a. Aprender sobre las tecnologías Aspire y NXT.
b. Analizar el proceso actual que comprende de una serie de tareas tantos manuales como automatizados para comprender las necesidades del cliente.
c. Proponer un diseño alternativo al proceso para la plataforma Aspire.
d. Desarrollar y documentar la propuesta de desarrollo.
e. Implementar el proceso en el servidor de producción para ser utilizado por el cliente.

1.6 Alcance del sistema

El sistema automatizado abarca todas las tareas que se realizan en el proceso existente en Perl. Se desarrollaron los módulos necesarios para llevar a cabo el procesamiento de texto debido además de la implementación de una interfaz web que permite que el cliente ejecute el proceso de actualización, crear respaldos y ver los reportes por sí mismo. De esta manera otorgarle más independencia y brindarle un mejor servicio al cliente.

Capítulo 2

Análisis del Proceso en Perl

2.1 Descripción General

FAST NXT Online Server es una suite electrónica que permite publicar, almacenar, recopilar, asegurar y distribuir contenido en línea, con búsquedas rápidas y navegación intuitiva. Aldrich Bonnefin & Moore utiliza esta herramienta para publicar su contenido por medio del Internet.

Inicialmente los documentos del cliente vienen en formato de Microsoft Word 2003 (.doc) y en formato Portable Document Format (.pdf). El proceso de actualización se basa en compilar estos manuales proveídos, indexar el contenido y generar un único archivo de colección propio de Rocket NXT (.nxt), para ser publicado en el servidor de producción donde los clientes podrán acceder al contenido.

El administrador del contenido[1] de Aldrich, copia al servidor vía FTP los archivos que desea actualizar. Y por medio de un correo indica a cuál manual corresponden esos documentos. El encargado en soporte, crea un respaldo de los archivos de la colección actual antes de copiar los nuevos documentos dentro de la colección. Luego debe abrir los archivos con títulos de manuales y cambiar la fecha de actualización, incluyendo al lado del título el mes y el año. Asimismo revisar que no hace falta efectuar cambios sobre los títulos de los apéndices. Al finalizar esta revisión, se ejecuta el script de Perl "aldrich.pl".

Este script realiza todo el procesamiento de texto, convierte los documentos de Word a formato HTML, edita texto por medio de expresiones regulares, convierte las notas de pie en eventos de java script, incluso invoca la ejecución de otros scripts como "makefile.pl", "linkcheck.pl" y "linkreport.pl".

[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 procesados los documentos, se construye la colección de NXT con el *ccbuilder*, esta colección es trasladada al servidor de pruebas donde es experimentado por el cliente. Y después de pasar por los procesos de aseguramiento de calidad y está libre de errores, se traslada al servidor de producción donde los subscritores a Aldrich podrán accederlo.

2.2 Frecuencia

Aldrich Bonnefin & Moore actualiza su contenido una vez al mes. Usualmente envían sus solitudes de cambios cada segundo lunes del mes.

2.3 Volumen de datos

La colección está compuesta por alrededor de 400 archivos. Estos ocupan aproximadamente 300 MB de espacio en disco.

2.4 Flujo general

La Figura 2.1 muestra las entradas, las salidas y la secuencia de tareas para el proceso de actualización de contenidos para Aldrich.

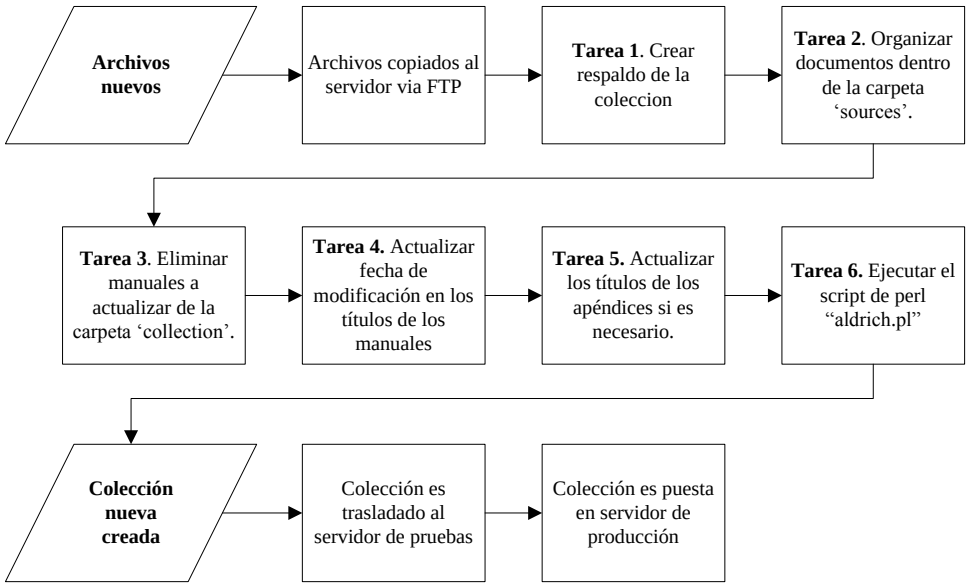

Figura 2.1 Flujo del proceso

2.5 Flujo detallado

Una de las tareas principales del script de "aldrich.pl" es el procesamiento de texto, que consiste en buscar patrones de texto y sustituirlos por otros valores. En la figura 2.2 se muestran los aspectos generales de este script.

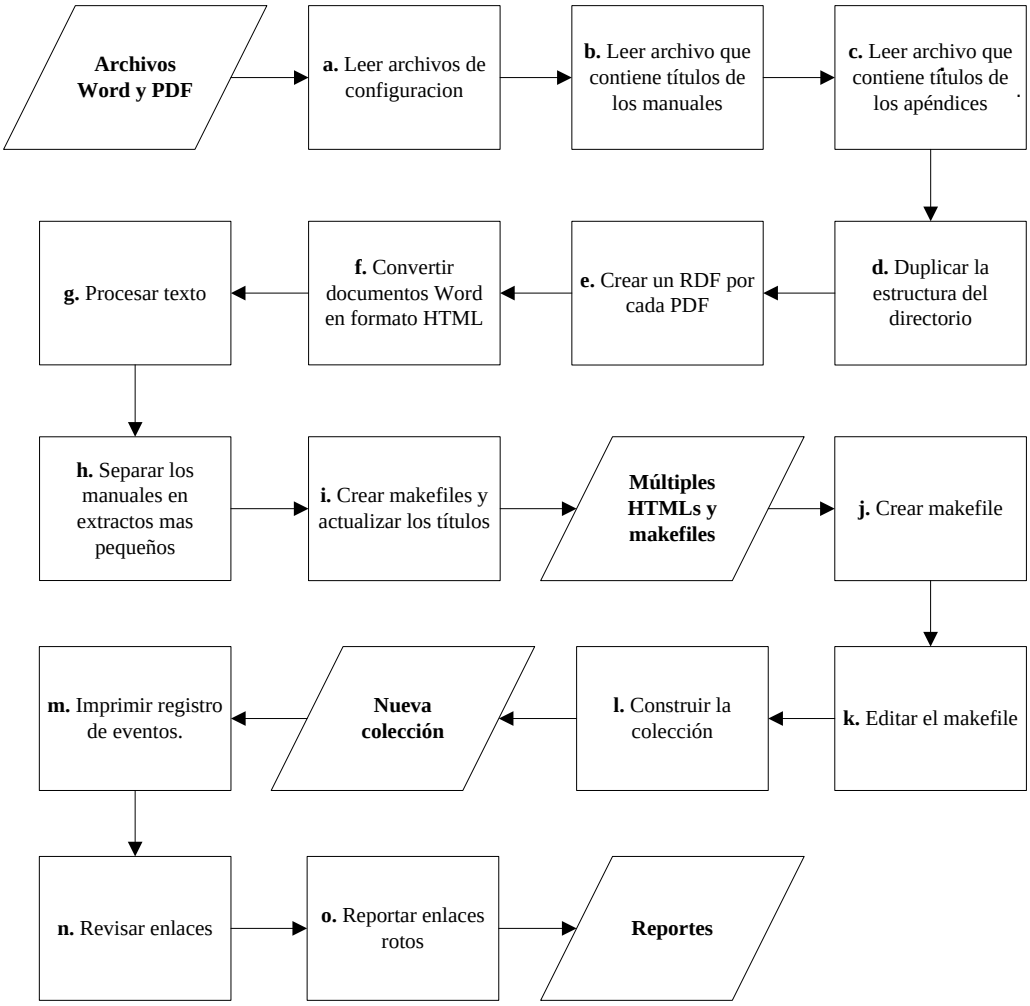

Figura 2.2 Flujo del script

Capítulo 3

Diseño de Componentes en Aspire

3.1 Requerimientos

La siguiente sección describe los requerimientos del sistema y cómo se esperaba que fueran migrados en Aspire para llevar a cabo la misma tarea.

1. *Identificar cambios en la estructura del directorio.*

   En el proceso en Perl se comparan las fechas de modificación de los archivos y se procesan únicamente aquellos con una fecha más reciente. Esto no era necesario en Aspire debido a que existe un componente conocido como *file-system-feeder*[1] que identifica los documentos a procesar al comparar el *directory snapshot*[2] actual con el último generado. Asimismo indica dentro de su *Aspire document*[3] si el documento ha sido agregado, actualizado o eliminado.

2. *Duplicar la estructura del directorio.*

   En el servidor se manejan dos directorios, una con los archivos fuentes originales y otro sobre el cual trabaja el proceso. Esto era necesario para que el cliente no tuviera acceso a los archivos generados por el proceso. En Aspire, se decidió que los documentos procesados fueran almacenados en otra ruta, de esta forma los documentos originales quedarían intactos.

   El convertidor de formatos permite generar su salida en otro directorio sobre el cual el resto de los procesos trabajaría, sin embargo luego se decidió crear un componente que se encargara de duplicar la estructura para poder eliminar los documentos borrados.

[1] Componente en Aspire que monitorea los cambios de un directorio.
[2] Archivos XML que contienen información sobre el contenido de un directorio específico.
[3] XML en memoria que contiene información cada proceso.

3. *Identificar el tipo de documento.*

   Aldrich posee los siguientes tipos de documentos: portadas, apéndices, tablas de autoridades, manuales y PDFs. Cada tipo de documento se procesa de forma distinta. Se diseñó un componente en Aspire que identificaría los tipos de documentos según su ruta de archivo. El archivo de configuración permitiría el ingreso de expresiones regulares, y si la ruta coincidiera con la regla, el proceso sería re-direccionado a otro serie de etapas para ser procesado correctamente.

4. *Asegurar que algunos archivos han terminado su procesamiento antes de continuar.*

   Se presentan dos casos en los cuales existe dependencia entre archivos: a) las tablas de autoridades deben ser procesadas antes que sus manuales, dado que estos últimos requieren de valores procesados de su respectiva tabla, y b) todos los documentos deben de haber terminado de procesar antes de crear el makefile para generar la colección.

   Aspire trabaja en un ambiente de múltiple hilos, en el cual los procesos son ejecutados aleatoriamente e inclusive simultáneamente en algunos casos. Para poseer un control sobre estos hilos, se decidió crear una barrera, en la cual si los procesos llegaban a este componente serían puestos en pausa. Y cuando hubiesen llegado todos los procesos a este componente, los hilos retomarían todos los procesos o únicamente el último proceso dependiendo de la configuración.

5. *Mantener datos en memoria para ser accedidos por distintos procesos.*

   Los títulos de los manuales y apéndices son utilizados para editar los makefiles individuales y el makefile que recompila a todos. Este componente en Aspire estaría encargado de leer estos valores y sería exportado para que otros componentes pudiesen accederlo por medio de una interfaz.

6. *Convertir archivos MS Word 2003 en formato HTML.*

   Perl utiliza un módulo COM que crea una interfaz a MS Office para abrir el archivo y guardarlo en formato HTML. Este proceso se migraría a Java para permitir la extensión de funcionalidad si se quisiera agregar más convertidores. De esta manera el componente será reutilizable. Para conversiones de Word a HTML se utiliza JACOB[1].

7. *Sustitución de texto.*

   El procesamiento de los documentos consiste principalmente de buscar por un patrón de texto y sustituirlo por otro valor. Se realiza una gran cantidad de sustituciones utilizando expresiones regulares para convertir el HTML generado legible por el *builder* del NXT.

   Java posee un paquete para manejar expresiones regulares. Este componente migrado a Aspire recibiría un archivo con las reglas de reemplazo y estas mismas también podrían ser digitadas dentro del archivo de configuración.

8. *Creación de makefile.*

   Se construye un makefile según la estructura de un directorio. En Perl se maneja como texto plano, pero Java tiene la ventaja de poseer paquetes que permiten la fácil manipulación de XML en memoria.

9. *Ejecutar comandos del sistema.*

   El *builder* de NXT es un ejecutable que se invoca por medio de la línea de comando o *command prompt*. El componente desarrollado en Aspire recibiría comandos y parámetros por medio del archivo de configuración. Los parámetros podrían ser referencias a valores dentro del *Aspire document*. Este componente se usaría para invocar el *ccbuilder* y el *winrar* para crear la colección y los respaldos respectivamente.

[1] Para más información, ver http://www.danadler.com/jacob/

3.2 Pipeline principal

Basados en los requerimientos identificados, se desarrolló el pipeline mostrado en la siguiente figura.

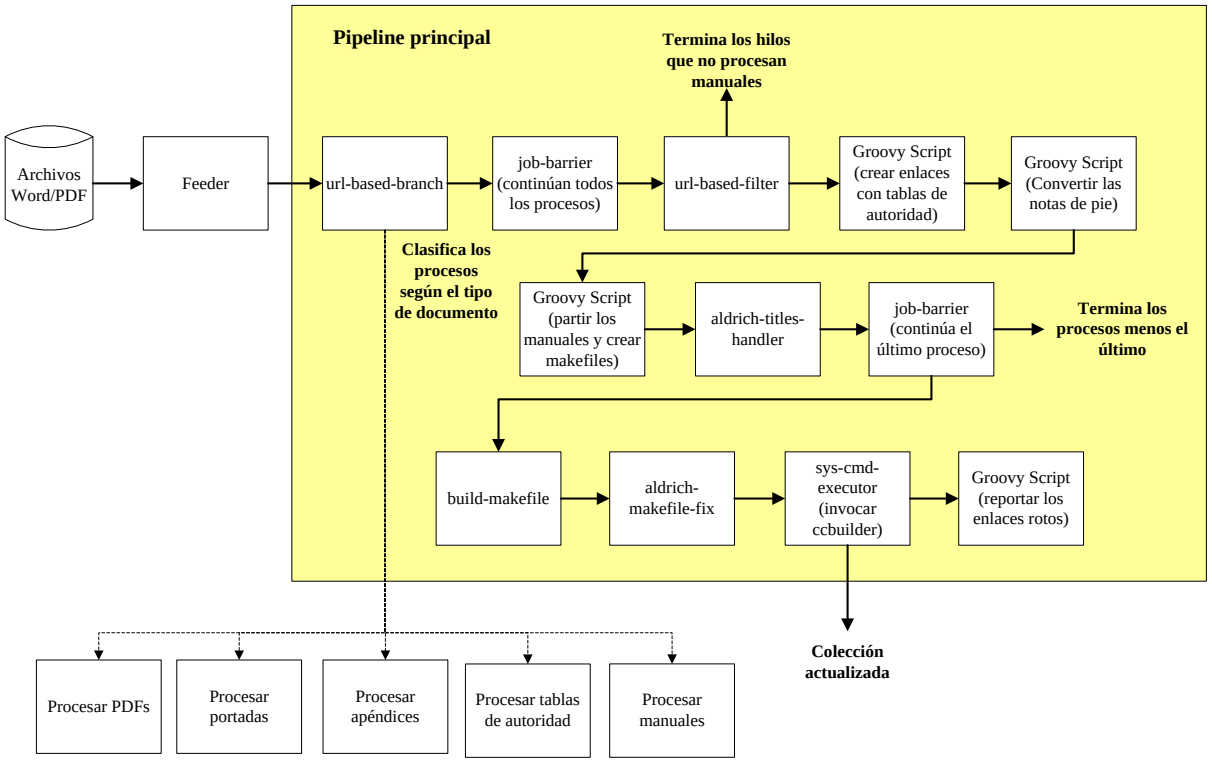

Figura 3.1 Diseño del pipeline principal

Los documentos de la colección serían publicados por el feeder al pipeline. El primer componente, identificaría los tipos de documentos por medio de la ruta del archivo. Cada tipo de documento sería re-direccionado al pipeline correspondiente para ser procesado por componentes más específicos. Y al terminar de procesarse en sus pipelines respectivos, estos procesos llegarían a la primera barrera, donde se esperan los procesos restantes. Una vez que haya llegado la cantidad de procesos publicados por el feeder, los procesos serían ingresados nuevamente a la fila de tareas para ser retomadas para continuar con el resto del pipeline.

El siguiente componente se encargaría de finalizar los procesos que no tienen referencia a un manual, dado que el resto del procedimiento corresponde únicamente a este tipo de documentos. Estos últimos componentes no fueron puestos dentro del pipeline que procesa únicamente manuales, dado que existe una dependencia entre estas y las tablas de autoridades. Por esta razón, la implementación de la barrera nos asegura que todas las tablas de autoridades han terminado de ser procesados.

Los tres componentes que continúan consisten de Groovy scripts. Este procesamiento se decidió implementar por medio de scripts porque las tareas que realizan son muy específicas al cliente Aldrich Bonnefin & Moore, por lo cual la probabilidad de reutilizar estos componentes es prácticamente nula.

Al finalizar de enlazar los manuales con su respectiva tabla de autoridad y de convertir los pie de notas en objetos Java script, estos serían divididos en extractos de HTML más pequeños. Y mientras se realiza esto, se iría creando simultáneamente un makefile individual para cada manual para indicar a qué título pertenece cada extracto. Este archivo generado luego pasaría por un componente que actualiza los títulos de los extractos para indicar a qué parte del manual pertenece. Estos valores se obtendrían mediante la lectura de dos archivos de títulos cuyos valores serían mantenidos en memoria para poder ser accedido luego por otro componente.

La siguiente barrera se aseguraría de que todos los manuales han terminado de procesar. Cada proceso que llega esta etapa es terminado menos uno. Este último tendrá la responsabilidad de generar un makefile leyendo el directorio fuente, editarlo para ingresarle los atributos y títulos apropiados, invocar el *ccbuilder* para construir la colección y revisar los enlaces y reportar aquellos que no tienen un destino válido.

En la tabla 3.1 se muestra un resumen de la funcionalidad de los componentes que conforman el pipeline principal.

Tabla 3.1 Descripción de los componentes del pipeline principal

COMPONENTE	DESCRIPCIÓN
url-based-branch	Re-direcciona un proceso a otro componente según las reglas indicadas en la configuración.
job-barrier	Espera por todos los procesos y permite que uno o todos los hilos retomen los procesos para continuar en el pipeline.
url-based-filter	Termina los procesos si la ruta del archivo coincide con una expresión regular especificada en la configuración.
anchorToa (Groovy)	Crea puntos de destino dentro del manual para que los enlaces en las tablas de autoridad sean válidos.
convertFootnotes (Groovy)	Convierte las notas de pie en objetos de Java script.
aldrich-titles-handler	Mantiene los títulos de manuales y apéndices en memoria además de actualizar los títulos de los makefiles individuales.
build-makefile	Genera un makefile nuevo a partir de un directorio.
aldrich-makefile-fix	Edita los elementos y atributos del makefile generado.
sys-cmd-executor	Invoca el *ccbuilder* por medio de la línea de comando.
reportLinks (Groovy)	Se encarga de revisar y reportar aquellos enlaces que no son válidos.

3.3 Pipelines secundarios

Los siguientes pipelines son los destinos a los cuales los documentos son re-direccionados según su tipo para recibir un procesamiento específico.

A continuación se despliega una figura general de los pipelines secundarios.

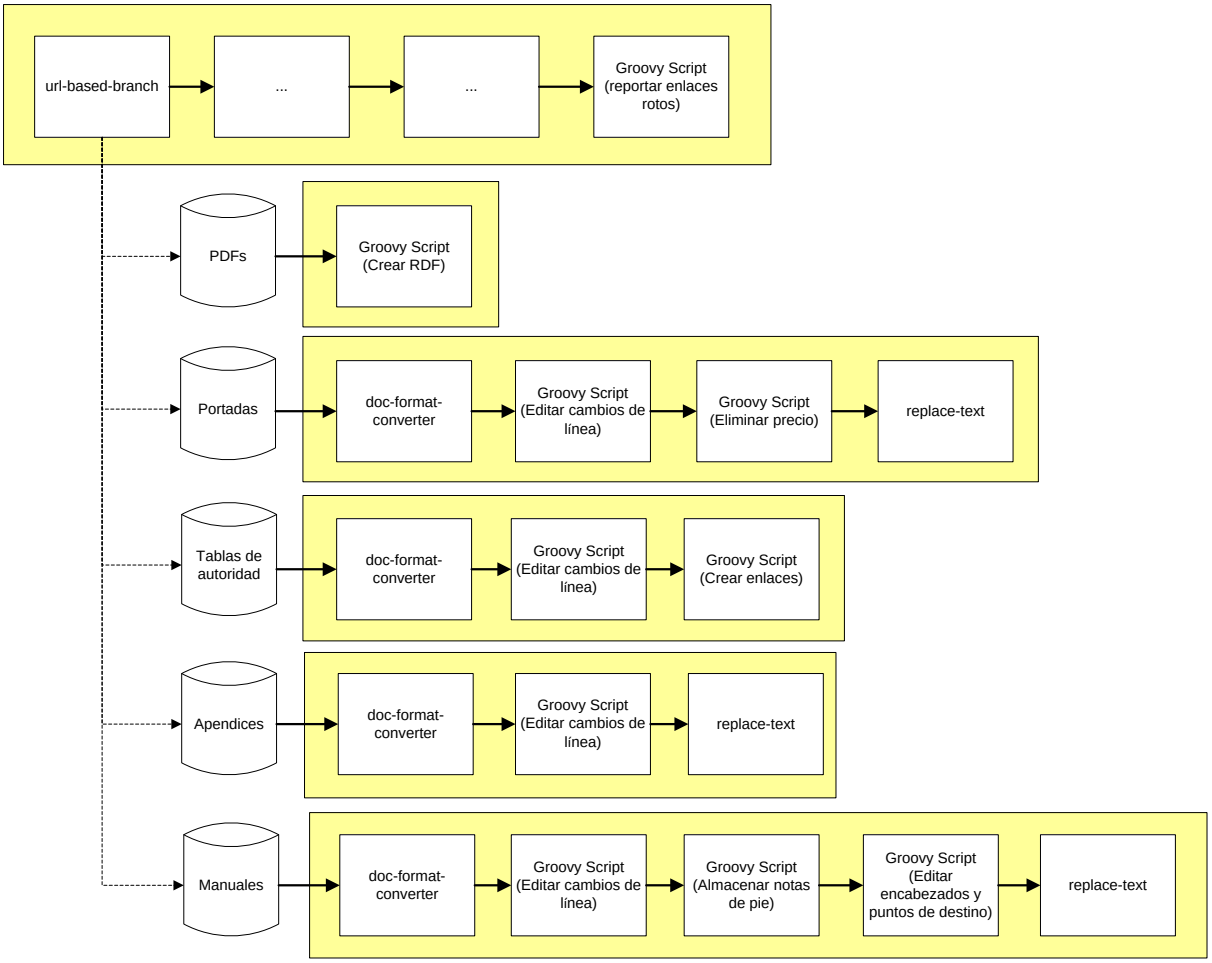

Figura 3.2 Diseño de los pipelines secundarios

Podemos observar que cada tipo de documento posee su propio pipeline. El pipeline encargado de procesar los PDFs consiste únicamente de un Groovy que creará un RDF para cada PDF. En cambio los demás documentos serán convertidos en HTML y serán procesados a mayor escala.

Se resume de forma general en la siguiente tabla la funcionalidad de cada componente de los pipelines secundarios.

Tabla 3.2 Descripción de componentes de los pipelines secundarios

COMPONENTE	DESCRIPCIÓN
createRdf (Groovy)	Copia los contenidos de "template.rdf" y reemplaza el número de manual y la letra del apéndice.
doc-format-converter	Convierte documentos Word en formato HTML en otro destino para que los clientes no tengan acceso a estos archivos.
editLineFeeds (Groovy)	Elimina los cambios de líneas y se insertan al inicio de encabezados y notas de pie.
removePricing (Groovy)	Elimina el precio de las portadas.
linkToa (Groovy)	Crea enlaces por cada valor dentro de la tabla de autoridad.
replace-text	Busca por un patrón de texto y lo reemplaza por otro valor.
obtainFootnotes (Groovy)	Almacena en el *Aspire document* las notas de pie encontradas.
prepareMaster (Groovy)	Edita los encabezados e inserta puntos de destino.

Capítulo 4

Solución Implementada

4.1 Implementación

Durante la etapa de desarrollo de componentes, surgieron elementos que no se tomaron en cuenta inicialmente, por lo cual se requirieron cambios no muy drásticos sobre el diseño.

En esta sección se describe el modelo implementado y la justificación de los cambios. Asimismo con una descripción más detallada de cada componente.

4.2 Pipeline principal

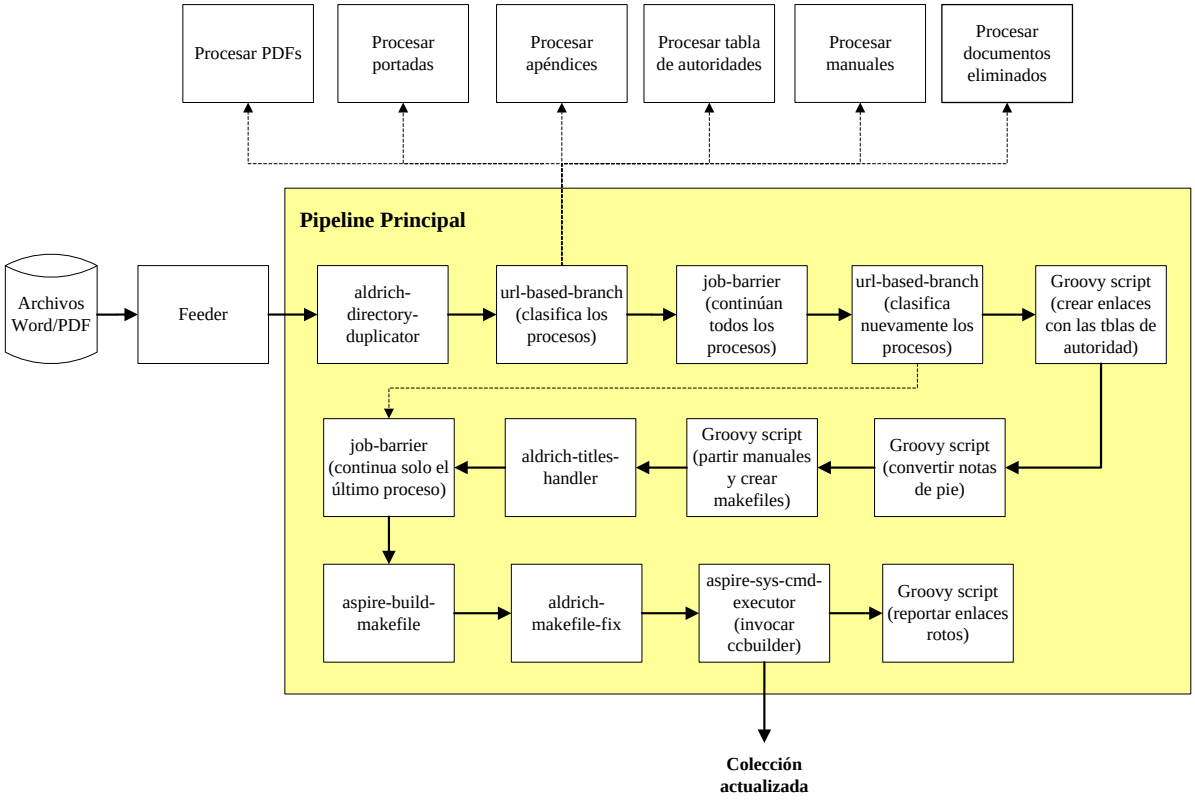

Figura 4.1 Pipeline principal implementado

Los cambios que se efectuaron sobre el diseño inicial fueron los siguientes:

a. Inicialmente para evitar que el cliente tuviera acceso a los documentos generados por el proceso, el componente encargado de convertir los documentos a formatos HTML, iba a generar estos documentos en otra ruta destino. Sin embargo, dado que la colección de Aldrich posee una estructura de directorios compleja era mejor crear un componente que se encargara en duplicar esta estructura. De esta forma ambos módulos, el convertidor de formatos y el duplicador de estructuras, cumplirían una función lo suficientemente genérica para permitir su reutilización en un futuro. Por esta razón, se incorporó un nuevo componente que se encarga específicamente de duplicar la estructura existente.

b. Dado que los componentes definidos después de la primera barrera procesan únicamente los manuales, se planeaba terminar los procesos que tenían referencia a los demás tipos de documentos, es decir, los procesos encargados de procesar portadas, tablas de autoridades, PDFs y apéndices eran finalizados ya que habían completado su procesamiento.

   La idea era liberar estos datos de la memoria, pero no se había contemplado el caso en el cual el cliente quisiera actualizar únicamente alguna portada, tabla de autoridad, apéndice o PDF. La nueva colección con estas actualizaciones no se generaría nunca porque estos procesos serían finalizados en la etapa del *url-based-filter*. Por lo tanto, en vez de terminar estos procesos en este punto, estos serían re-direccionados a la segunda barrera. En el cual igualmente serían finalizados por excepción de una, que llevaría a cabo el resto del procedimiento para generar la nueva colección.

c. Otro punto que no se tomó en cuenta fue que el feeder publica documentos borrados. Si se borra un documento del directorio, el feeder publica una tarea con la referencia al documento eliminado. Este documento no podrá ser procesado dado que ya no existe, por lo cual debe ser re-direccionado a la segunda barrera para el caso en que las actualizaciones de la colección consisten únicamente de documentos eliminados. De esta forma se generará nuevamente la colección, pero sin los archivos borrados.

   Este cambio en el pipeline principal implicó una adición en el diseño de los pipelines secundarios. Cuando un documento fue procesado previamente, se generaron archivos a partir de ellos. De cada apéndice, tabla de autoridad, manual y portada se crea un HTML y por cada PDF se crea un RDF. Por lo cual si un archivo Word es borrado, también deben ser borrados estos archivos asociados a los documentos originales. En el caso de los manuales, que en un punto dado son partidos en extractos más pequeños, todos estos extractos deben ser borrados si el manual es eliminado. Así en el momento que se crea el makefile, no se vayan a crear referencias a estos documentos.

   Aunque el componente duplicador de estructura, elimina el documento publicado como borrado, no elimina los documentos asociados a él. Por esta razón, se agregó otro pipeline secundario en el cual un Groovy Script es encargado de eliminar estos documentos. Como sabemos que por cada manual existe únicamente, una portada, una tabla de autoridad y el manual. Las carpetas que contienen estos serán eliminadas para eliminar todos aquellos documentos derivadas de ellas. En cambio, cada manual posee una cantidad variada de apéndices, por tanto esta carpeta no es eliminada, sino solo su HTML. Y si es un PDF el documento eliminado, se elimina su RDF.

   De esta forma se reflejará en la carpeta sobre el cual se ejecuta el proceso el mismo contenido que posee el directorio con la colección actual.

4.3 Pipelines secundarios

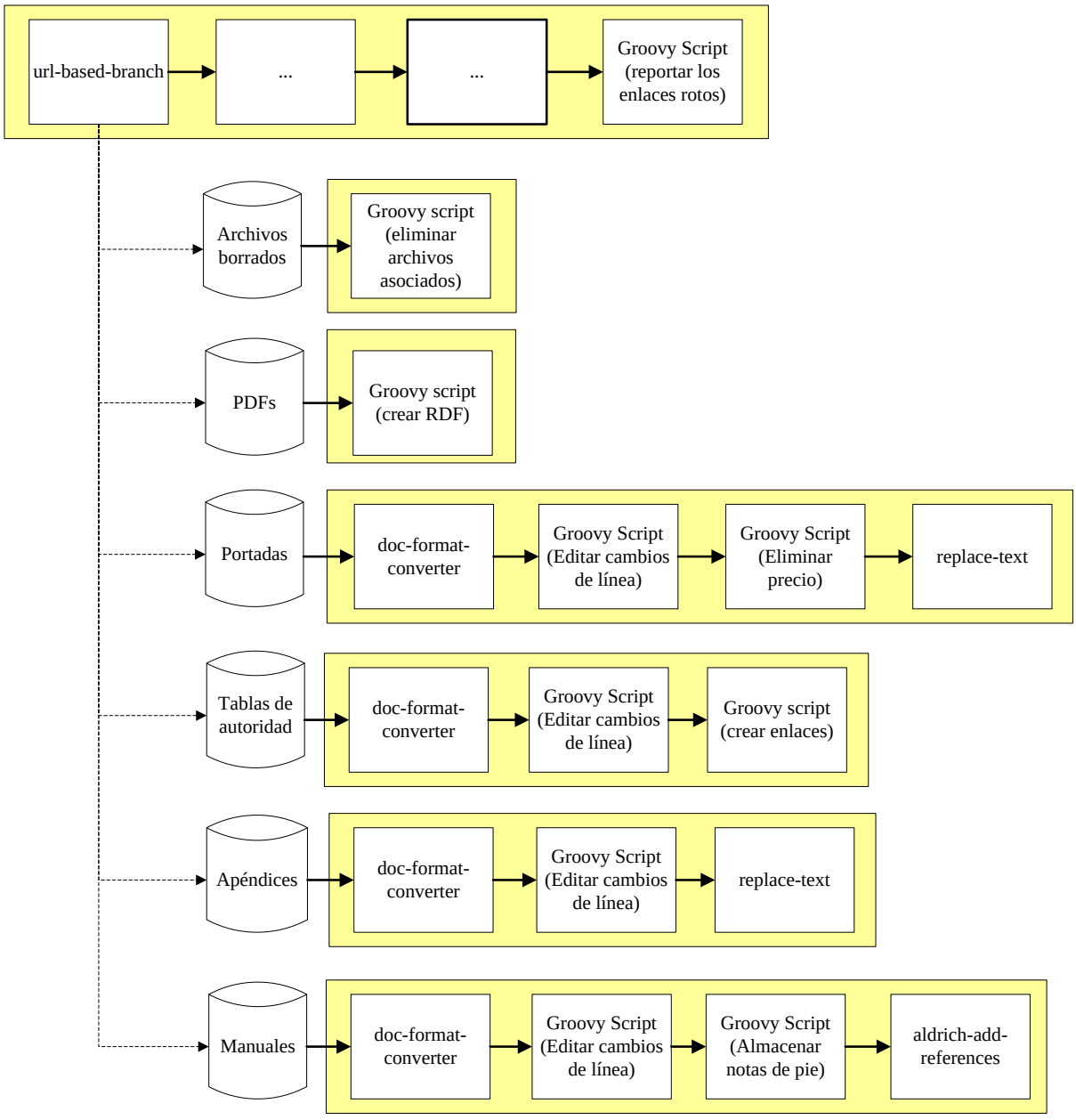

Figura 4.2 Pipelines secundarios implementados

Además de la incorporación de un pipeline que procese los archivos que fueron eliminados, se generaron otros cambios menores en el pipeline encargado de procesar los manuales.

Inicialmente se iba a desarrollar un Groovy Script que editaría los encabezados, que insertaría puntos de destino, y este script prepararía el documento para ser procesado por el componente encargado de reemplazar texto. Sin embargo al implementar el diseño se presentaron un par de problemas:

a. El componente *replace-text* en este caso busca por texto al cual debe editar para agregarle enlaces. Estos enlaces tienen referencias a valores obtenidos dinámicamente. Mientras se recorre línea por línea el manual, se van almacenando valores para indicar la parte y sección del manual en el que se encuentra. Estos valores son utilizados para los mismos enlaces y si se mapean todos estos valores dentro del *Aspire document* requeriría de mucha memoria y además requeriría de muchos cambios sobre las expresiones regulares existentes para poder identificar luego cual valor debe ser utilizado. Consecuentemente se decidió unificar el componente de sustitución de texto junto con la etapa previa, creando así el componente *aldrich-add-references*.

b. Al probar el nuevo componente, se observó que la duración en esta etapa era excesivo. Dado que se aplican más de mil quinientas expresiones regulares complejas por cada línea de un manual, y cada manual consiste de aproximadamente ochocientos páginas, el tiempo de procesamiento incrementó exponencialmente. Para resolver este problema se consideró optimizar las expresiones regulares, pero modificar una cantidad tan grande de expresiones regulares es una tarea tediosa y propensa a errores. Consiguientemente se tuvo que buscar otra alternativa para reducir la aplicación de expresiones regulares por lo cual se decidió agrupar las expresiones por una palabra o frase clave. Esta clave consistiría de una expresión regular y si esta coincidía con la línea leída, se le aplicaría una lista de expresiones regulares. En el caso contrario, se ahorraría la búsqueda de algunas expresiones regulares innecesarias.

4.4 Componentes

A continuación se describen todos los componentes desarrollados.

4.4.1 Feeder

Un feeder es un componente responsable de identificar los documentos disponibles para ser procesados. Obtienen información, usualmente meta-data del documento, y envían el documento como un nuevo proceso dentro el manejador de componentes: el pipeline manager. Aspire posee distintos tipos de feeders de los cuales monitorean distintos tipos de fuentes. Sin embargo, un nuevo feeder tuvo que ser desarrollado para cumplir con los requerimientos.

En la siguiente tabla se muestran detalles del componente.

Tabla 4.1 Descripción del componente start-all-feeder

STARTALL FILE FEEDER	
Entrada	Los documentos modificados del directorio monitoreado.
Salida	Un objeto *Aspire document* con los elementos <url> y <fetchUrl> que contiene la ruta del documento y un atributo "action" que indica si el documento ha sido agregado, actualizado o eliminado, publicado al pipeline manager.
Descripción	Este feeder es una variación de un feeder existente, el system-file-feeder. Este componente publica todos los documentos modificados y les asigna un id a cada uno. Esto permite al feeder publicar al pipeline manager la cantidad total de documentos que deben ser procesados para permitir el funcionamiento correcto de la barrera.

4.4.2 Stages

Una serie de estos componentes componen un pipeline, o línea de ensamblaje por el cual pasa el proceso. A continuación se describen los componentes que procesan los documentos.

Tabla 4.2 Descripción del componente aspire-url-based-branch

CLASIFICADOR DE PROCESOS	
Entrada	El objeto *Aspire document* y los valores del archivo de configuración.
Salida	Proceso re-direccionado a otra etapa o pipeline.
Descripción	Este componente está configurado para comparar un elemento o atributo del *Aspire document* con un conjunto de expresiones regulares especificados en el archivo de configuración. Si el valor coincide con la regla, entonces el proceso se envía ya sea a otra etapa dentro del pipeline o a otro pipeline distinto.

Tabla 4.3 Descripción del componente aspire-directory-duplicator

DUPLICADOR DE DIRECTORIOS	
Entrada	El directorio fuente, el directorio destino, la referencia al documento y el atributo "action" que indica si fue agregado, actualizado o eliminado.
Salida	Una copia o la eliminación del documento en el directorio destino.
Descripción	Se ejecuta la acción definida por el atributo sobre el mismo documento en el directorio destino.

Tabla 4.4 Descripción del componente aspire-doc-format-converter

CONVERTIDOR DE FORMATOS	
Entrada	El elemento <url> del *Aspire document*, que hace referencia al documento, el formato a convertir y la ruta destino para la salida.
Salida	Documento en el nuevo formato con el elemento <url> actualizado con su nuevo ruta y nombre de archivo, y el mapeo de meta datos.
Descripción	Este componente utiliza JACOB para crear un puente COM al MS Office para convertir los documentos de Word a HTML. Está diseñado para escoger dinámicamente el convertidor apropiado para efectuar la conversión. También extrae meta data del documento original y lo mapea al *Aspire document*.

Tabla 4.5 Descripción del componente aspire-replace-text

RE-EMPLAZADOR DE TEXTO	
Entrada	Referencia del documento en el <url> y la lista de reglas del archivo de configuración.
Salida	Documento procesado.
Descripción	Este componente aplica una serie de búsquedas y reemplazos de texto en el contenido del archivo. La configuración especifica los patrones a buscar y los valores a reemplazar. Los reemplazos pueden contener referencias a valores del *Aspire document*.

Tabla 4.6 Descripción del componente aspire-job-barrier

BARRERA DE PROCESOS	
Entrada	Un hilo que lleva a cabo un proceso.
Salida	Dependiendo de la configuración permite todos los procesos o únicamente el último proceso continuar en el pipeline.
Descripción	Los procesos que llegan al componente son puestos en pausa hasta que los procesos restantes hayan llegado a este componente también. El objetivo de la barrera es sincronizar los procesos. Este componente trabaja junto el feeder que le comunica a través del pipeline manager la cantidad de documentos que publicó. De esta forma podrá saber la cantidad de procesos que debe esperar para saber si ya llegaron todos los procesos o no a este determinado punto. Cada vez que un proceso es terminado ya sea porque finalizó o porque se produjo un error, el contador es decrementado. Sin embargo, si el proceso de actualización es detenido, todos los procesos deben de reiniciar. Este componente solo asegura que cuando se ejecuta el proceso de actualización, los documentos procesados han llegado hasta determinado punto.

Tabla 4.7 Descripción del componente aspire-sys-cmd-executor

EJECUCIÓN DE LÍNEA DE COMANDO	
Entrada	Nombre del comando y sus parámetros. Los parámetros pueden ser referencias a elementos dentro del *Aspire document*.
Salida	Salida del comando ejecutado si ocurre un error.
Descripción	Este componente ejecuta una línea de comando con los parámetros definidos en el archivo de configuración. El componente es independiente de la plataforma, pero las utilidades que se invocan pueden no serlo, por tanto el comando a ejecutar depende de la máquina que está ejecutando Aspire.

Tabla 4.8 Descripción del componente aldrich-titles-handler

MANEJADOR DE TÍTULOS	
Entrada	Makefiles individuales y los archivos con los títulos de apéndices y manuales.
Salida	Makefiles con títulos actualizados.
Descripción	Este componente es muy especifico a Aldrich Bonnefin & Moore. Lee los títulos de los archivos para mantenerlos en memoria y permiten ser accedidos por otros componentes por medio de una interfaz. También procesa los makefiles de cada manual actualizando los títulos de cada elemento.

Tabla 4.9 Descripción del componente aspire-build-makefile

CONSTRUIR MAKEFILE	
Entrada	Una referencia a un directorio del cual se va a crear el makefile o un makefile existente.
Salida	Nuevo makefile con referencias a los documentos de la colección.
Descripción	Si no hay un makefile existente se construye uno nuevo a partir de la plantilla y de un directorio. Si existe, será actualizado.

Tabla 4.10 Descripción del componente aldrich-makefile-fix

ARREGLAR MAKEFILE	
Entrada	Referencia al makefile creado dentro del elemento <url>.
Salida	Makefile arreglado para ser leído por el *ccbuilder*.
Descripción	Este componente también fue desarrollado específicamente para Aldrich Bonnefin & Moore. Edita los valores de algunos atributos y reemplaza los archivos del manual por el makefile individual generado previamente. Este proceso accede al componente manejador de títulos para agregar los títulos de los manuales a los elementos del makefile que son portadas.

Tabla 4.11 Descripción del componente aldrich-add-references

AGREGAR REFERENCIAS	
Entrada	Referencia al manual y archive de texto con la reglas de sustitución.
Salida	Manuales con enlaces y puntos de destino incorporados.
Descripción	Este componente específico a Aldrich convierte algunas etiquetas de encabezados en etiquetas de párrafos y etiquetas de ancla para construir puntos de destino a los enlaces. Tiene una funcionalidad muy similar al componente de *replace-text*, a excepción de que las expresiones están agrupadas por otra expresión regular. Y si se cumple ésta condición, entonces se aplican las expresiones agrupadas. Los valores de reemplazo contienen referencias a valores dentro de *Aspire document* y de variables que se encuentran en memoria durante el procesamiento.

4.4.3 Groovy scripts

Groovy facilita la creación de componentes sin tener que generar código Java y un archivo JAR. En esta sección se describen los groovy scripts desarrollados para llevar a cabo el procesamiento requerido para la colección de Aldrich Bonnefin & Moore.

Tabla 4.12 Descripción de los Groovy scripts

ENTRADA	SALIDA	DESCRIPCIÓN
Manuales en formato HTML	Manuales HTML con puntos de destino	**anchorToa.script** Inserta etiquetas con valores de su respectiva tabla de autoridad para que se puedan crear enlaces.

ENTRADA	SALIDA	DESCRIPCIÓN
Manuales en formato HTML	*Aspire document* con las notas de pie	**obtainFootnotes.script** Este script busca por las notas de pie y los almacena en su *Aspire document* para ser utilizados más adelante.
Manuales en formato HTML	Manuales HTML con objetos Java script	**convertFootnotes.script** Convierte las pie de notas en eventos *mouse-overs* de Java script utilizando el texto de las notas de pie almacenados previamente en el *Aspire document*.
Tabla de autoridad en formato HTML	Tablas de autoridad con enlaces NXT	**linkToa.script** Este script crea un enlace NXT para cada valor dentro de la tabla de autoridad.
Manual en formato HTML	Múltiples archivos HTML y un makefile	**splitMaster.script** Los manuales son partidos en extractos más pequeños generando varios archivos HTML. Mientras se realiza esto, se construye un makefile que hace referencia a estos documentos.
Archivos HTML	Un reporte	**reportLinks.script** Revisa los enlaces creados y reporta aquellos que no tienen un destino válido.

ENTRADA	SALIDA	DESCRIPCIÓN
Portadas en formato HTML	Portadas sin precio	**removePricing.script** Remueve el precio de la portada de un manual.
Archivos HTML	HTML con cambios de línea.	**editLineFeeds.script** Elimina todos los cambios de línea y las inserta al inicio de cada encabezado y nota de pie. También inserta dentro del *Aspire document* el numero de manual al que corresponde el archivo.

4.5 Diagramas de clases

A continuación se muestran los diagramas de clases de los componentes reutilizables con las clases nuevas creadas o modificadas, los atributos y operaciones implementados.

Start All File Feeder

Este feeder hereda de las clases *SimpleFeederImpl* y *FileFeederImpl*. Se le agregó a la clase *SimpleFeederImpl* un nuevo método que establece la cantidad total de procesos publicados. Este método se agrega en todas las clases hasta llegar a la clase de *PipelineManagerImpl* en donde el componente de barrera accederá para consultar la cantidad de procesos a esperar.

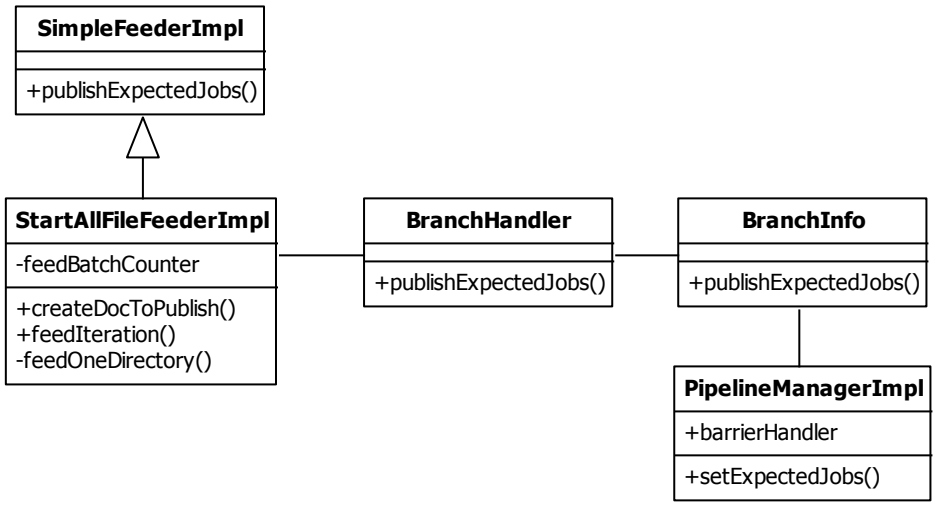

Figura 4.3 Diagrama de clases del start-all-file-feeder

Duplicador de estructura de directorios

Este componente copia los archivos al directorio destino si el documento publicado ha sido agregado o actualizado. Si ha sido borrado, entonces es eliminado del directorio destino.

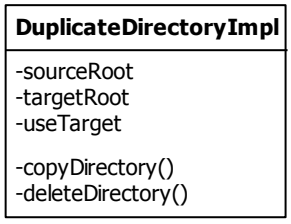

Figura 4.4 Diagrama de clases del componente directory-duplicator

Convertidor de formatos

El componente recibe un formato y/o una ruta destino. La clase *ConverterFactory* determina cuál convertidor utilizar dependiendo de la entrada y formato para ejecutar la conversión requerida.

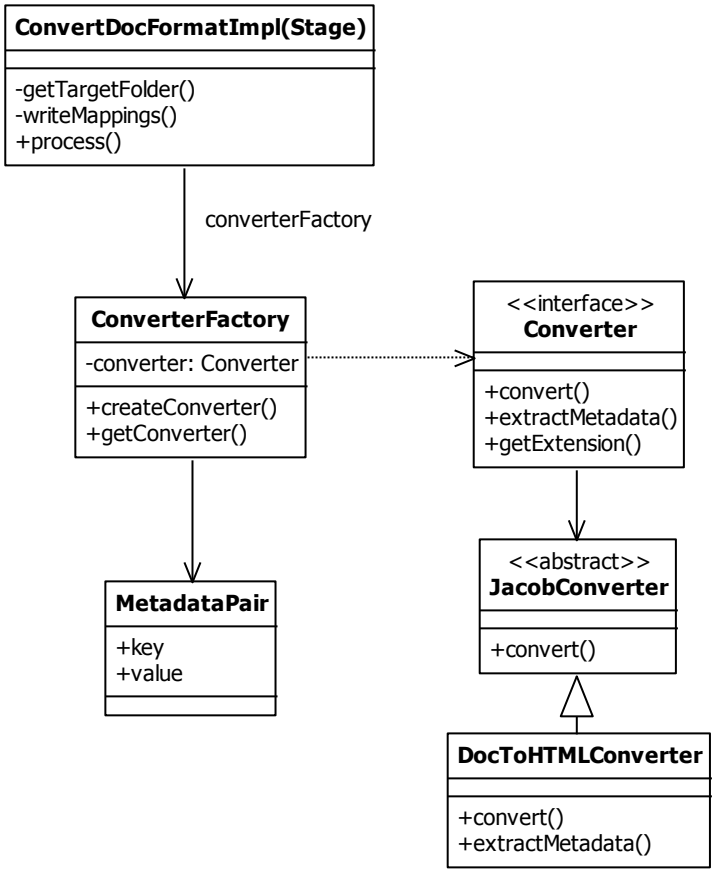

Figura 4.5 Diagrama de clases del componente doc-format-converter

Sustitución de texto

Este componente recibe un listado de reglas de sustitución por medio de la configuración o un archivo externo para ser aplicado sobre el documento. Por naturaleza este componente tiene gran potencial en causar un cuello de botella. Dependiendo del tamaño del documento, la cantidad de documentos y complejidad de las expresiones regulares, el rendimiento puede verse afectado.

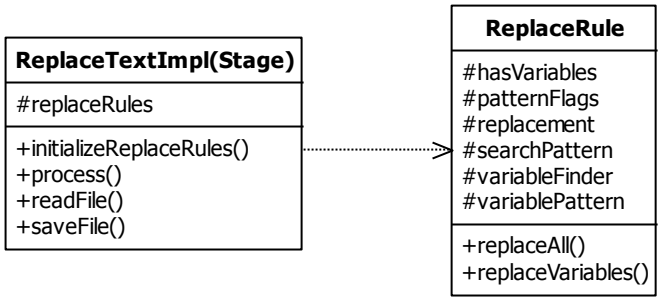

Figura 4.6 Diagrama de clases del componente replace-text

Clasificador de procesos

Este componente permite comparar valores ya sea un texto o un atributo del *Aspire document* contra un listado de expresiones regulares. Si cumple las reglas, se re-direcciona el proceso al evento especificado en el archivo de configuración.

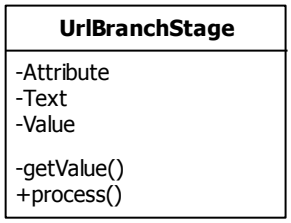

Figura 4.7 Diagrama de clases del componente url-based-branch

Construcción del makefile

La clase *Makefile* es creada para implementar la funcionalidad de crear uno nuevo o de actualizar uno existente. Esta nueva clase provee métodos para agregar o eliminar documentos del makefile, crearlo a partir de un directorio y en base a una plantilla.

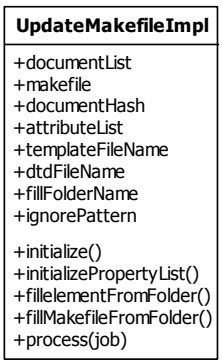

Figura 4.8 Diagrama de clases del componente build-makefile

Ejecución de la línea de comando

Ejecuta comandos con parámetros estáticos u obtenidos del *Aspire document*.

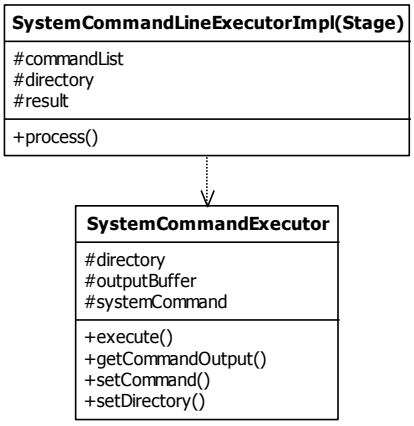

Figura 4.9 Diagrama de clases del componente sys-cmd-executor

Barrera de procesos

Cuando un proceso alcanza esta etapa, será puesto en pausa hasta que todos los procesos publicados alcancen este punto. Este componente simula el patrón de barrera para sincronizar los hilos, pero aplicado a nivel de procesos. No hay intervención con los hilos. La clase *BarrierHandler* es agregada para administrar la sincronización y la información de la barrera. Una instancia de esta clase es creada por el método nuevo *initBarrier()* de la clase *PipelineManagerImpl*. Una vez inicializada el controlador de barreras, cuando se invoca el método *await()* se actualiza la bandera *isPaused* en verdadero, para que el manejador de procesos indique en la lista de procesos, que ha llegado esta tarea a la barrera. Esto insertará la tarea en *arrivedJobList* y finalizará su ejecución liberando el hilo para que procese otra tarea. Cuando llegue el último proceso a ejecutar *await(),* dependiendo de la configuración publicará o no de nuevo las tareas al pipeline manager para terminar la ejecución del pipeline.

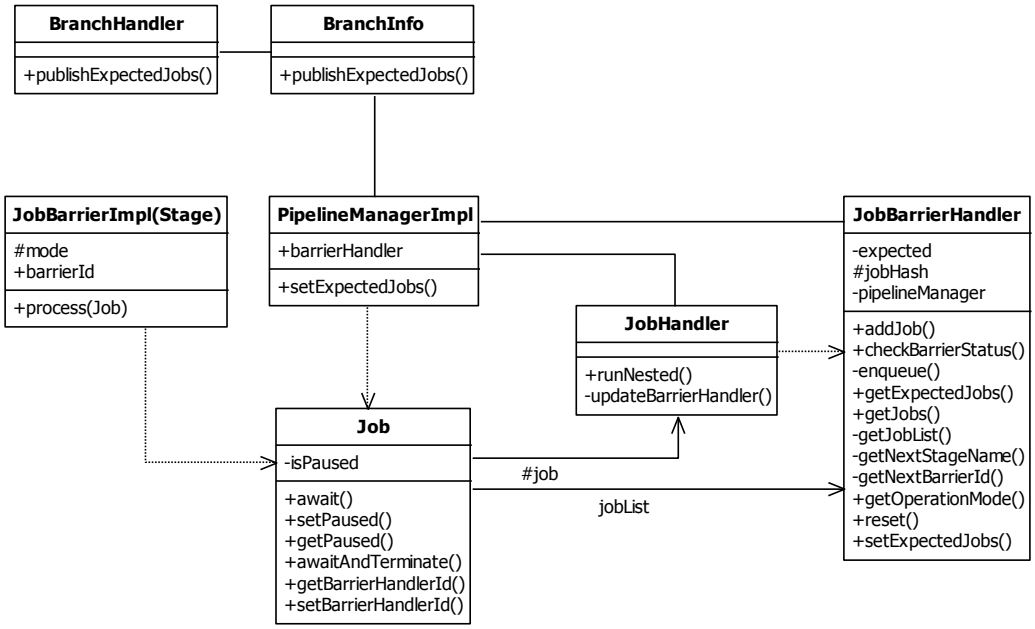

Figura 4.10 Diagrama de clases del componente job-barrier

4.6 Interfaz de usuario

Se desarrolló una página web que permite el cliente tener control sobre el proceso de actualización. La interfaz permite:

a. Acceso al FTP para ingresar los archivos de la colección.
b. Iniciar y detener el proceso de actualización.
c. Crear un respaldo de los documentos en la colección existente.
d. Ver un reporte de eventos.
e. Ver un reporte de enlaces rotos.

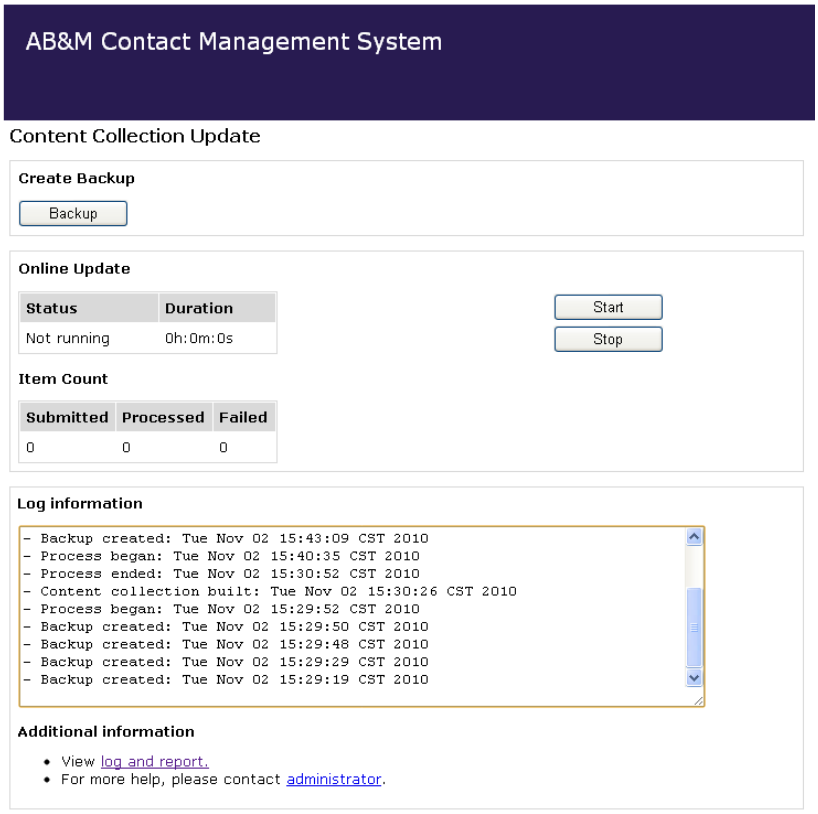

Figura 4.11 Interfaz de usuario

En la figura 4.11 podemos ver la interfaz web que se comunica con el servidor de Aspire. Esta página contiene información sobre el proceso de actualización, reportes y los botones de control. La siguiente tabla despliega el significado de cada campo y sus posibles valores.

Tabla 4.13 Campos de la interfaz web

CAMPO	DESCRIPCIÓN	POSIBLE VALORES
Status	Indica el estado del proceso.	*Running*: Ejecutando *Not Running*: Detenido *Stopping*: Intentado detener el proceso *Unavailable*: La conexión a Aspire no se pudo establecer.
Submitted	Muestra la cantidad de documentos publicados por el feeder.	Entero positivo.
Processed	Indica la cantidad de documentos que han terminado de procesar exitosamente.	Entero positivo.
Failed	Indica la cantidad de documentos que han fallado durante el proceso.	Entero positivo.
Duration	El tiempo desde que inicio el proceso.	Horas, minutos y segundos.

Todos los valores son reiniciados cada vez que se ejecuta el proceso de actualización.

Una vez presionado el botón de “Stop”, esto detendrá al feeder, sin embargo, se seguirán procesando los documentos que ya fueron publicados.

El cuadro de texto despliega un resumen de los eventos ocurridos. Se pueden observar los reportes con mayor detalle en un enlace en la sección de información adicional de la página.

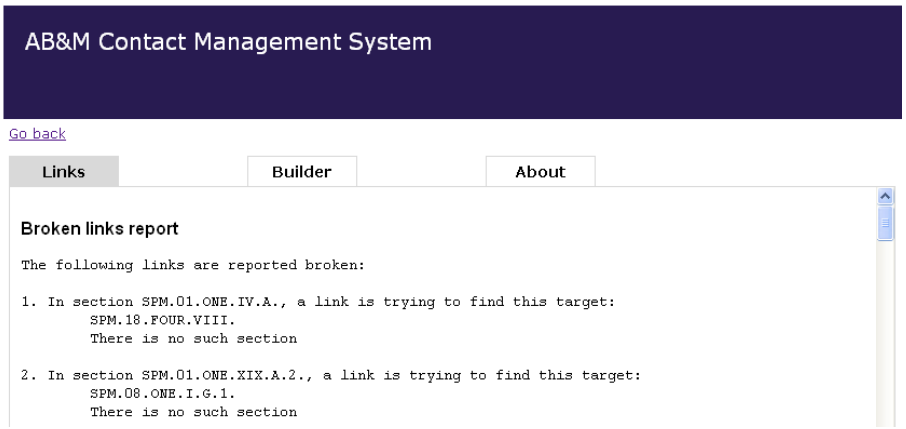

Figura 4.12 Reportes

4.7 Diagrama de componentes

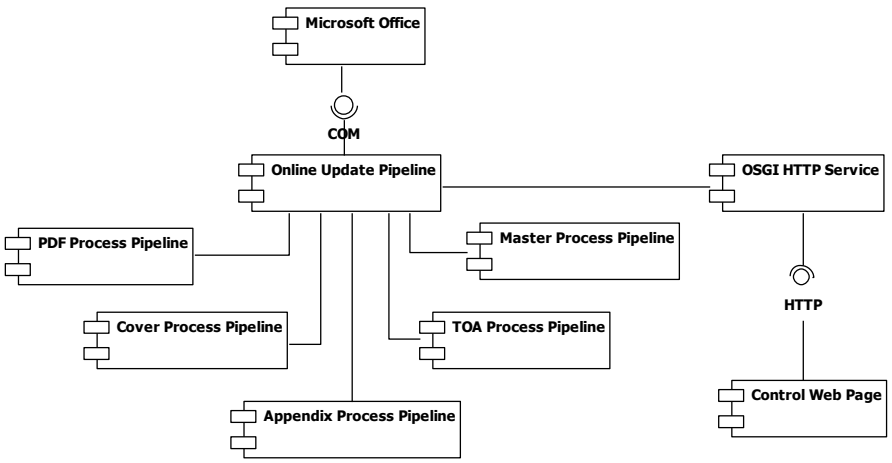

Figura 4.13 Diagrama de componentes

Capítulo 5

Análisis de Resultados

5.1 Entregables

Además de los módulos desarrollados, se definieron tres entregables para este proyecto de práctica:

a. El primer informe consistía de un análisis del proceso existente en Perl. Este documento describía ampliamente el proceso de actualización desarrollada por la antigua Info Solutions.
b. El segundo informe presentaba una propuesta de desarrollo. Esta fase consiste en el diseño. Se describe cómo se van a desarrollar los componentes, cómo se van a integrar y qué funcionalidades van a llevar a cabo. Adjunto van diagramas de clases y de componentes.
c. El tercer informe describe la solución implementada, los cambios realizados y la justificación de las mismas. Se incluye además las conclusiones, recomendaciones y descripción sobre la experiencia adquirida.

5.2 Pruebas

Se realizaron mediciones de tiempo de la duración de ambos procesos de actualización en una máquina con las siguientes características:

- Windows XP Professional 2002, Service Pack 3
- 1.83 GHz, 3.25 GB de RAM
- Disco duro de 80 GB

Los resultados fueron los siguientes:

Variables	Perl	Aspire
Utilizando Office 2007	25 minutos	Aproximadamente 16 minutos
Utilizando Office 2003	23 minutos	Aproximadamente 14 minutos

Al utilizar Office 2003, se reduce el tiempo en ambos procesos, pero el ahorro es mínimo dado que esta es la duración de la actualización de toda la colección, es decir, este es la duración del proceso en el peor de los casos.

Variables	Aspire
Sin agrupar las expresiones regulares, Office 2007	Aproximadamente 60 minutos
Agrupando las expresiones regulares, Office 2007	Aproximadamente 16 minutos

Podemos observar que el tiempo se redujo notablemente. Inicialmente cuando no se implementó la agrupación de expresiones regulares para el componente que agrega los enlaces NXT en los documentos, el proceso completo duraba más de 60 minutos, mientras que al utilizar la agrupación y aplicando únicamente una expresiones selectas, se aceleró este proceso en un 80%. Aunque este procesamiento se implementó de una forma distinta a la existente, presenta el mismo resultado con mayor eficiencia.

5.3 Conclusiones

Cuando se inició con el proyecto, se desconocía sobre las tecnologías utilizadas para manejar grandes contenidos de información. Mediante este proyecto se tuvo la oportunidad de aprender sobre FAST NXT y las ventajas que se obtienen al utilizar herramientas como esta, tales como la indexación, la navegación intuitiva, los mecanismos de búsquedas y las tablas de contenido. Asimismo se aprendió sobre la arquitectura de Aspire, el cual resalta la utilidad de poseer componentes reutilizables y configurables, mayor procesamiento al utilizar múltiples hilos, y cómo un diseño elaborado permite fácilmente la extensión de una funcionalidad.

La migración del proceso de conversión de Perl a Aspire se realizó exitosamente. Los entregables especificados se desarrollaron y se entregaron en las fechas de aceptación. Y como podemos observar en los resultados de las pruebas realizadas, mediante la automatización del proceso de actualización en Aspire, se logró reducir un 40% del tiempo invertido en la ejecución del mismo. El objetivo planteado se alcanzó y la nueva solución presenta una mejora.

5.4 Recomendaciones

Algunas recomendaciones para reproducir o mejorar este sistema se mencionan a continuación:

1. Poseer conocimiento sobre funcionamiento de hilos para desarrollar componentes *thread-safe*.
2. Crear los componentes de la forma más genérica posible para permitir su reusabilidad.
3. Tener el código en módulos facilita su mantenimiento.
4. Documentar en paralelo al desarrollo.
5. Definir una convención para crear enlaces para simplificar y evitar ambigüedades durante el procesamiento textual.

6. Utilizar en los archivos de configuración los valores Ascii o secuencias de escape para los caracteres especiales.
7. Comprender la sintaxis y los cuantificadores existentes para denotar las expresiones regulares.
8. Optimizar las expresiones regulares para mejorar el rendimiento. Se logra al:
    a. Reducir el *backtracking*.
    b. Agrupar expresiones similares.
    c. Agrupar las expresiones más comunes al utilizar *OR*s.
    d. Guardar la referencia de un grupo únicamente solo cuando se necesita.
9. Agregar las rutas de las aplicaciones en la variable *PATH* del sistema para poder invocar la aplicación desde el *aspire-sys-cmd-executor* únicamente con el nombre. Después de agregado, se debe reiniciar la maquina.
10. Agregar en las variables del sistema *JAVA_OPTS=Xmx256* para incrementar la memoria alocada para los Groovy Scripts de ser necesario.
11. Utilizar un API de OpenOffice para no depender de la plataforma de Windows.
12. Investigar posibles alternativas para efectuar la búsqueda de patrones y reemplazos para mejorar el rendimiento del componente *aspire-replace-text*.
13. Proveer soluciones que otorguen al cliente mayor control sobre el proceso y reducir la dependencia.

5.5 Bibliografía

1. Adler, D. (2004). *The Jacob Project*. Recuperado de *The Java Com Bridge*: http://www.danadler.com/jacob/
2. Fast NXT. *Online Server Documentation*. Recuperado de *Tour the Default Site*: http://www.amlegal.com/nxt/gateway.dll/np/000intro/002defaultsitetour.htm?f=templates$fn=document-frame.htm$3.0
3. Friedl, J. *Perldoc.perl.org*. Recuperado de *Mastering Regular Expressions*: http://perldoc.perl.org/perl.html
4. Rocket Software. (2010). *Rocket Software*. Recuperado de http://www.rocketsoftware.com/
5. Search Technologies. (2010). *Wiki*. Recuperado de *Aspire Documentation*: http://wiki.searchtechnologies.com/
6. The Apache Software Foundation. (2004). *Jakarta ORO*. Recuperado de *Demonstration Applet*: http://jakarta.apache.org/oro/demo.html

Apéndices

1. Diccionario de términos

Aspire	Un procesador de documentos de alto-rendimiento en un ambiente multi-hilo que permite el manejo de procesos.
CCbuilder	El *content collection builder* es una utilidad que permite crear una colección a partir de un sistema de archivos. Los archivos de la colección son definidos por un makefile. Esta herramienta es provista por Rocket NXT.
Colección NXT	Documentos estructurados e indexados. Las colecciones de NXT poseen tres piezas de información: tabla de contenidos, el índice y copia del contenido.
Java script	Una implementación del lenguaje EXMAScript para acceder objetos computacionales dentro de un ambiente. Usualmente utilizados en páginas web para crear interfaces más dinámicos.
Makefile	Básicamente es un script que guía la utilidad make de UNIX para escoger los archivos que deben ser compilados y enlazados.

2. Wiki de Search Technologies

La empresa Search Technologies posee un wiki en el cual Aspire está documentado ampliamente. Describe la funcionalidad de Aspire y cómo debe ser configurado. Se recomienda consultar el wiki para obtener mayor comprensión, pero se requiere de credenciales para poder accederlo, es decir, se requiere de un usuario y contraseña para acceder la página: https://wiki.searchtechnologies.com/mediawiki/

3. Índice de Figuras

4. Índice de Tablas

Anexos

Se adjuntan las configuraciones del pipeline principal y secundarios.