UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

TITULO: Procesamiento por Lotes en el Sistema Bancario Cubano.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS.

AUTOR

Osvaldo Rodríguez García
orgarcia@estudiantes.uci.cu

TUTOR

Ing. Adolfo Miguel Iglesias Chaviano
aiglesias@uci.cu

Ciudad de la Habana, 2008 - 2009
Año 50 de la Revolución

Declaración de Autoría

Declaro ser autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año

_____.

Osvaldo Rodríguez García

_____
Firma del Autor.

Ing. Adolfo Miguel Iglesias Chaviano

_____
Firma del Tutor.

Datos de contacto

Autor:
Osvaldo Rodríguez García
Estudiante Universitario

Cursante del 5to año de la Facultad 4 de la Universidad de las Ciencias Informáticas, con un promedio actual de 4.48 puntos.
Correo electrónico: orgarcia@estudiantes.uci.cu

Tutor:
Ing. Adolfo Miguel Iglesias Chaviano
Profesor Adiestrado

Graduado en Julio del 2007 de Ingeniero en Ciencias Informáticas en la Universidad de las Ciencias Informáticas con Título de Oro y Promedio General de 5.09 puntos. Seleccionado Premio "Julio Antonio Mella" en la primera graduación de dicha Universidad. Ha impartido las asignaturas de Programación I y II, Ingeniería de Software I y II y el curso de Patrones de Diseño en la Universidad de las Ciencias Informáticas. Formó parte del equipo que elaboró dicho curso. Miembro de la Reserva del Comandante en Jefe. Participó como desarrollador en el proyecto SAFRE y en un Sistema de encuesta sobre temas de Nefrología. Analista y desarrollador del Subsistema Sala Situacional del Sistema de Gestión Penitenciaria para la República de Venezuela. Ha elaborado e impartido curso de capacitación al proyecto relacionado con la modernización del Sistema Bancario Cubano. Actualmente Arquitecto Principal del proyecto relacionado con la modernización del Sistema Bancario Cubano.
Correo electrónico: aiglesias@uci.cu

Centro:
Universidad de las Ciencias Informáticas (UCI).
Carretera a San Antonio de los Baños, km 2 ½, Boyeros, Ciudad de La Habana, Cuba.

Índice

Contenido

Ilustraciones

Tablas

Agradecimientos

A mis padres por estar a mi lado en todo momento apoyándome con sus consejos y por mostrarme el camino correcto para seguir adelante, gracias a ustedes hoy soy Ingeniero, gracias a ustedes siempre seré un hombre de bien, y gracias a ustedes existo.

A mi hermana, por cuidarme tanto y por inspirarme con su esfuerzo, siempre me tendrás y siempre te voy a necesitar.

A mis abuelos, por ser mis maestros en la asignatura de la vida, por enseñarme con sus experiencias, por cultivarme con su amor, y por ayudarme a realizar mis sueños.

A toda mi familia, por el aliento, el ánimo y el cariño incondicional que siempre me han profesado.

A Yaima por ser mi compañera insomne en las largas noches de desvelo, por ser mi pareja y amiga durante los últimos 4 años, y por llenar mi vida de amor.

A mis amigos, por la confianza, el amor y la amistad que me han transmitido en el calor de un abrazo, la humedad de un beso, o en las palabras escritas por culpa de la distancia.

A mis compañeros de estudio por ser tan tolerantes conmigo, les agradezco los buenos y los malos momentos, y les aseguro que por todo lo que de ustedes he podido aprender ...nunca los voy a olvidar.

A mi tutor Adolfo, por alumbrar con la luz de sus conocimientos los últimos pasos de mí andar de estudiante. A los profesores del tribunal, por sus sabios consejos y sus críticas constructivas. A todos aquellos profesores que a lo largo de mi carrera han sembrado en mí la semilla del saber.

Al Ingeniero en Informática Dariel Bombino Revuelta por aportar sus conocimientos, y su apoyo incansable.

A Fidel…Raúl…y a la Revolución, por engendrar las ideas que hoy hacen posible mi realización como Ingeniero en Ciencias Informáticas.

A todos aquellos que de alguna manera hicieron su aporte en mi formación…

¡Muchas Gracias!

Dedicatoria

Quiero dedicarle el resultado de este trabajo, a toda mi familia, quienes tanto a mi lado como en la distancia han apostado sus esfuerzos y han depositado su esperanza en verme convertido en un profesional.

Resumen

El uso de diferentes técnicas para procesar información se ha incrementado exponencialmente a lo largo del desarrollo de la informática, producto de que cada día los Software desarrollados se enfrentan a la creciente demanda de datos pendientes por procesar, y a una mayor cantidad de procesos en espera de informatizarse. El Sistema Bancario Cubano que se desarrolla actualmente en la Universidad de Ciencias Informáticas, se encuentra inmerso en la necesidad de procesar grandes cantidades de información en el menor tiempo posible. En este trabajo se presenta la investigación realizada, acerca de los avances logrados en el Procesamiento por Lotes (Procesamiento Batch), así como los sistemas existentes que de alguna manera implementen esta técnica, y las tecnologías disponibles que la utilicen. Se expone de manera resumida, mediante argumentos, conceptos, y definiciones; las diferentes aristas que brinda la utilidad del uso de esta técnica, y el alcance de la misma en el procesamiento de grandes cantidades de información. Finalmente se brinda la propuesta de una arquitectura a seguir, y/o desarrollar, así como los recursos y gestiones necesarias para lograrla; con el objetivo de poder procesar todo el cúmulo de información con la que se trabaja en el Sistema Bancario Cubano, que actualmente se encuentra en desarrollo.

Introducción

En la Universidad de las Ciencias Informáticas (UCI), entre otros proyectos, se desarrolla un software para el Sistema Bancario Cubano.

No es nada eventual que un sistema computacional (o sea un software), se enfrente en algún momento de su desarrollo a la imponente necesidad de procesar grandes cantidades de información, en ocasiones chocando con las exigencias de que esto se haga en el menor tiempo posible, o que se aprovechen al máximo las tecnologías. Esta **situación problemática** y sus antecedentes, son las que a lo largo del desarrollo de la Informática, han dado lugar al Procesamiento por Lotes (Procesamiento Batch[1]).

Los sistemas por lotes son el mecanismo más tradicional y antiguo de ejecutar tareas. Se introdujeron alrededor de 1956 para aumentar la capacidad de proceso de los programas. En la actualidad se puede encontrar este mecanismo en la mayoría de las aplicaciones, entre los más conocidos por su uso cotidiano se encuentran: R-Project, Adobe Photoshop, TMPG Encode (Batch Encode Tool), así como su influencia en los Sistemas Operativos, los Framework, Lenguajes de Programación y Sistemas de Gestión de Base de Datos. Realmente casi cualquier programa puede ejecutar tareas en modo Batch, siempre y cuando permita especificarse los distintos pasos de ejecución que debe seguir, o las entradas de usuario a partir de un script[2].

Hasta este momento, y con el objetivo de inducir una idea general sobre el Procesamiento por Lote, se expone como concepto general que: El procesamiento por lotes, consiste en ejecutar un programa o una parte de este, sin el control ni la supervisión directa del usuario. Generalmente, este tipo de ejecución se utiliza en tareas repetitivas sobre grandes conjuntos de información, ya que sería tedioso y propenso a errores realizarlo manualmente. Estos programas suelen especificar su funcionamiento mediante scripts o guiones (procedimientos), en los que se indica qué se quiere ejecutar, e incluso se puede definir qué tipo de recursos necesita reservar.

El Sistema Bancario Cubano, enfrenta el siguiente **problema**: ¿Qué hacer con las grandes cantidades de información que debe procesar en un breve período de tiempo el sistema

[1] **1. Ref.** Referente al procesamiento por Lote en aplicaciones cuya versión se encuentra en Ingles.
[2] **2. Sinónimo.** guion, plantilla.

bancario?, pues no se cuenta actualmente con un mecanismo o una manera de procesar estas grandes cantidades de información sin la interacción del usuario en el menor tiempo posible.

Esto podría provocar grandes demoras, y numerosos gastos, en el procesamiento de datos, para prevenir este inconveniente se decide realizar un estudio con el **objetivo** de proponer un diseño arquitectónico capaz de procesar los grandes volúmenes de información que maneja el sistema bancario cubano, con el fin de facilitar una solución viable. Después de un análisis de varias técnicas de procesamiento de datos, se determina que el **objeto de estudio** será: los mecanismos de procesamiento Batch, siempre enfocado al que más se acople a las necesidades y exigencias del sistema, e incidiéndose directamente sobre el siguiente **campo de acción:** el área de procesamiento de datos del Sistema Bancario que actualmente se encuentra en desarrollo.

Este trabajo tiene como principal **idea a defender**:

Brindar una arquitectura a seguir y desarrollar, que permita procesar las grandes cantidades de información que maneja el Sistema Bancario Cubano que actualmente se está desarrollando.

Capítulo 1. Fundamentación Teórica

1. Introducción.

En este capítulo se brindará de manera resumida la información necesaria acerca del estado del arte y de los avances logrados en el tema del Procesamiento por Lotes (Procesamiento Batch), a lo largo del desarrollo de la Informática y hasta el momento actual en el que se desarrollará la investigación.

Los sistemas por lotes son el mecanismo más tradicional y antiguo de ejecutar tareas. Se introdujeron alrededor de 1956 para aumentar la capacidad de proceso de los programas, y aunque su utilización se ha visto resumida a resolver necesidades esenciales en determinados programas, lo cierto es que la aplicación de esta técnica de manera correcta brinda todo un universo de nuevas posibilidades en el procesamiento de datos.
En la actualidad este mecanismo se encuentra en la mayoría de las aplicaciones, entre las cuales constan: R-Project, Adobe Photoshop, TMPG Encode (Batch Encode Tool), entre otros de uso común para la mayoría de los usuarios, pues realmente, casi cualquier programa puede ejecutar en modo Batch, siempre y cuando pueda especificarse los distintos pasos de ejecución o las entradas de usuario que precisa su correcto funcionamiento, ya sea a través de scripts o cualquier forma de configuración.
El alcance de esta técnica ha ido creciendo exponencialmente, pues su aplicación se ve vinculada a todos los tipos de tecnologías, software, sistemas operativos, y plataformas existentes en la actualidad. Tal es así, que en el desarrollo de este capítulo se muestra de manera resumida, los orígenes del procesamiento por lotes desde su existencia en los sistemas operativos mediante los conocidos archivos (.bat), hasta su implantación en los más potentes gestores de base de datos, y los más actuales Frameworks de desarrollo de software.

A manera de concepto general e introductorio, entiéndase entonces que el procesamiento por lotes, consiste en ejecutar un programa o una parte de este, sin el control ni la supervisión directa del usuario. Generalmente, este tipo de ejecución se utiliza en tareas repetitivas sobre grandes conjuntos de información, ya que sería tedioso y propenso a errores realizarlo manualmente.

2. *Archivos de Procesamiento por Lote (.bat)*

2.1. Introducción

Los archivos de procesamiento por lotes, o más bien conocidos como .bat; son muy comunes en la mayoría de los sistemas operativos y aplicaciones ejecutables. Su función consiste en ejecutar una serie de tareas, así como proporcionar los parámetros y recursos necesarios para el correcto funcionamiento de las mismas, sin la necesidad de que el usuario intervenga, ni interactúe con el ordenador. Un ejemplo evidente, podría verse en el propio sistema operativo Windows XP, el cual cuenta en su carpeta raíz del sistema (C:\), con un archivo llamado AUTOEXEC.BAT, este tiene como función principal registrar las tareas programadas, configuradas previamente por el usuario con el fin de ejecutarlas al iniciarse el sistema de manera automática; es perfectamente editable desde cualquier editor de texto, aunque su correcto funcionamiento precisa un alto conocimiento de los comandos y funciones de consola.

Un archivo de procesamiento por lote proporciona una forma abreviada de ejecutar uno o varios mandatos del **MS−DOS**. Cuando se teclea solo el nombre de un archivo de procesamiento por lote, el archivo ejecuta cada línea como si se la estuvieran introduciendo desde el teclado.

Los archivos de procesamiento por lote pueden automatizar instrucciones largas o repetitivas. La posibilidad de cometer errores en la digitación[3] de un mandato se reduce, y las tareas largas se pueden comenzar y dejar que se ejecuten sin prestarles atención. La escritura de esta clase de archivos se puede concebir como una forma de programar **MS−DOS**.

Este tipo de archivo, figura dentro de la bibliografía existente en la Informática como la forma más conocida de procesamiento Batch, y aunque es cierto que tienen una gran aplicación simplificando los pasos necesarios para cargar la configuración de la mayoría de los sistemas operativos; su explotación de la técnica de procesamiento por lote deja mucho que desear, al resumirse solamente a dar órdenes de manera escrita al ordenador.

[3] **1. f.** Adiestramiento de las manos en la ejecución musical con ciertos instrumentos, especialmente los que tienen teclado. Referente a la habilidad practica de los dedos sobre el teclado. (Diccionario de la Real Academia: http://buscon.rae.es/draeI/).

2.2. Creación de Archivos de Procesamiento por Lote (.bat)

Se pueden crear archivos de procesamiento por lote utilizando **COPY CON**, cualquier procesador de palabras capaz de crear archivos de texto (la mayoría puede), o un editor de texto.
Nota: **COPY CON** no se recomienda para los ejemplos largos dada la inconveniencia para corregir errores de digitación[4].
Nota: Si se utiliza un procesador de palabras, se deberá utilizar el modo **ASCII** o "*no documento"*.
El modo normal de muchos procesadores de palabras almacena los caracteres tecleados en un código que es el **MS−DOS**. Si el programa de procesamiento de palabras no hace distinción entre *"documentos"* y "*no documento"*, deberemos utilizar el siguiente método para crear un archivo de procesamiento por lote.

- Digitar[5] un archivo de procesamiento por lote sencillo. Cada línea del archivo debe ser un solo mandato ejecutable del **MS−DOS**. Evitar el subrayado, las negritas u otro formato especial. Asegurarse de que no aparezcan los símbolos de cambio de línea u otros en la pantalla. Un archivo de procesamiento por lote no debe tener el mismo nombre raíz que el de un archivo de programa (un archivo que termine con **.COM** o **.EXE**) en el directorio en curso. También se pueden utilizar los marcadores de parámetros (%0 a %9), las variables de ambiente con el nombre de la variable entre signos de porciento (como **%COMSPEC%**), y los sub-mandatos de procesamiento por lote. Guardar el archivo con una extensión de tipo .bat; después intentar ejecutarlo en la línea de solicitud del **MS−DOS**.

2.3. Ejecución de un Archivo de Procesamiento por Lote (.bat)

Se puede ejecutar un archivo de procesamiento por lote introduciendo el nombre del archivo en la línea de solicitud del MS−DOS mediante la siguiente sintaxis:

---

[4] **1. f.** Adiestramiento de las manos en la ejecución de ciertos instrumentos, especialmente los que tienen teclado. Referente a la habilidad practica de los dedos sobre el teclado. (Diccionario de la Real Academia: http://buscon.rae.es/draeI/).

[5] **1. tr.** Chile, El Salv., Hond. y Ur. Incorporar datos a la computadora utilizando el teclado. **intr.** Chile, Hond. y Ur. Manejar los dedos con destreza, especialmente al hacer funcionar un instrumento provisto de teclas o cuerdas.

(um:)(trayectoria\)(nombre_archivo) (parámetros)

(um:) es el nombre de la unidad de disco que contiene al archivo de procesamiento por lote.

(trayectoria\) es la trayectoria del archivo de procesamiento por lote.

(nombre_archivo) es el nombre raíz del archivo de procesamiento por lote.

(parámetros) son los parámetros que utilizara el archivo de procesamiento por lote.

Nota: () Son totalmente innecesarios, su uso está limitado a encapsular la sintaxis para un mejor entendimiento de la misma.

2.4. Reglas para ejecutar archivos de Procesamiento por Lote (.bat)

Para invocar un archivo de procesamiento por lote, basta con teclear su nombre raíz. Por ejemplo, para invocar el archivo de procesamiento por lote **FREC.BAT**, digitar **FREC**, y después pulsar la tecla **ENTER**. **MS−DOS** ejecuta cada mandato en una línea a la vez, aunque solo reconoce un máximo de diez parámetros, se puede usar el sub-mandato **SHIFT** para superar esta limitación. Si el **MS−DOS** encuentra un sub-mandato de procesamiento por lote incorrectamente escrito, emite un mensaje de error de sintaxis, para después continuar con los mandatos restantes del archivo. Se puede detener la ejecución de un archivo de procesamiento por lote presionando **ctrl.−Break**. También se puede hacer que el **MS−DOS** ejecute un segundo archivo de procesamiento por lote inmediatamente después de que finalice el primero. Simplemente se debe introducir el nombre del segundo archivo como el último mandato del primer archivo, o se puede ejecutar un segundo archivo dentro del primer archivo y regresar a éste usando el subcomando **CALL**. Los sub-mandatos de procesamiento por lote son válidos solo para archivos de este tipo. No es posible, pues, ejecutar sub-mandatos de procesamiento por lote como mandatos normales del **MS−DOS**. No se puede redirigir la entrada o salida de un archivo de procesamiento por lote. Sin embargo, se puede usar redirección en las líneas dentro de un archivo de este tipo.

3. Aplicaciones del Procesamiento por Lote

3.1. Introducción

Actualmente la técnica de Procesamiento por Lote, es aplicada por la mayoría de los programas que se usan diariamente en la informática, en la mayoría de los casos está presente como una herramienta adjunta y su función se reduce a brindar la posibilidad de grabar una serie de pasos que permitan posteriormente hacer funcionar el software con el que se esté trabajando sin la interacción del usuario ni la entrada de datos, basado solamente en la configuración previamente realizada. A modo de ilustrar este enfoque que mayormente tiene el Procesamiento por Lote en aplicaciones de uso común, se selecciono como ejemplo el software Adobe Photoshop.
Esta técnica de procesamiento de información también es aplicada en el desarrollo de tecnologías para sistemas bancarios, a través de potentes herramientas, como son los Framework y los sistemas gestores de Bases de Datos. Como ejemplo ilustrativo se muestra el caso de la empresa “Modhelus”.

3.2. Procesamiento por Lote en aplicaciones de uso común

Con el objetivo de brindar como ejemplo el uso del procesamiento por lote en las aplicaciones comunes, se describirá como se aprovecha esta funcionalidad en Photoshop.

3.2.1. Procesamiento por Lote en Photoshop

El software propietario Adobe Photoshop perteneciente a la compañía Adobe, implementa en todas sus versiones, entre otras funcionalidades, la técnica de procesamiento por lotes, a través de su comando *Lote (Batch)*.
Esta aplicación como bien se conoce, concentra su gran utilidad en el procesamiento, la administración, y la edición de fotografías. Pero pocos conocen que su alcance llega hasta el

punto de ser capaz de procesar grandes cantidades de imágenes en un corto plazo de tiempo y sin la necesidad de que intervenga el usuario, con solo seguir una serie de pasos y haciendo uso del *Comando Lote*.

3.2.2. Utilizar el comando Lote en Photoshop

El comando Lote permite ejecutar una acción en una carpeta de archivos o subcarpetas. Si tiene una cámara digital o un escáner con un alimentador de documentos, también puede importar y procesar varias imágenes con una única acción.
Cuando procesa archivos por lotes, puede dejar todos los archivos abiertos, cerrar y guardar los cambios realizados en los archivos originales, o guardar las versiones modificadas de los archivos en una nueva ubicación (dejando los originales sin modificar). Si guarda los archivos procesados en una nueva ubicación, puede crear una nueva carpeta para los archivos procesados antes de iniciar el lote.

3.2.3. Pasos para procesar archivos mediante el comando Lote en Photoshop

1- Seleccione Archivo > Automatizar > Lote.
2- En Origen, seleccione un origen en el menú emergente:
   - Carpeta, para ejecutar la acción en archivos que ya se encuentren almacenados en el ordenador. Haga clic en Seleccionar para localizar y seleccionar la carpeta.
   - Importar, para importar y ejecutar la acción en imágenes desde una cámara digital, escáner o documento PDF.
   - Archivos abiertos, para ejecutar la acción en todos los archivos abiertos.
   - Explorador de archivos, para ejecutar la acción en los archivos seleccionados en el Explorador de archivos.
3- En Destino, Seleccione un destino para los archivos procesados.
4- Seleccione Ignorar comandos “Guardar como” de Acción si prefiere las instrucciones “Guardar como” del comando Lote a las instrucciones “Guardar como” de la Acción. Si

selecciona esta opción, la acción debe contener un comando “Guardar como”, ya que el comando Lote no guardará automáticamente los archivos de origen. Esto es útil para guardar documentos con opciones no disponibles en el comando Lote (como opciones de compresión JPEG o TIFF, etc.)

Nota: Independientemente de cómo grabe los pasos Guardar como de la acción (con o sin especificaciones de nombre de archivo), si esta opción está seleccionada, el archivo se guarda en la carpeta y el nombre de archivo en el comando Lote.

5- Seleccione una opción para procesar errores en el menú emergente Errores:

- Detener para buscar errores, para suspender el proceso hasta que se confirme el mensaje de error.
- Errores de registro a archivo, para grabar cada error en un archivo sin detener el proceso. Si los errores se registran en un archivo, aparece un mensaje después del proceso. Para revisar el archivo de error, ábralo con un editor de texto tras ejecutar el comando Lote.

Nota: Para realizar el proceso por lotes con varias acciones, cree una nueva acción que ejecute las demás acciones y, a continuación, procésela por lotes (puede anidar acciones dentro de acciones). Para procesar por lotes varias carpetas, cree alias en la carpeta para las otras carpetas que desee procesar y seleccione la opción Incluir todas las subcarpetas.

3.2.4. Ventajas del comando Lote en Photoshop

Este procedimiento le permite, por ejemplo, enfocar, redimensionar y guardar las imágenes como JPEG en las carpetas originales. Así como realizar cualquier tipo de acción posible, de la misma manera que si usted estuviera delante del ordenador, con la diferencia de que su interacción no es necesaria, si configuro correctamente cada uno de los pasos a seguir.

Reduce considerablemente el tiempo que tomaría procesar una gran cantidad de imágenes por los métodos convencionales, y gestiona los recursos del ordenador en función de un mejor rendimiento mientras se ejecuta el proceso por Lote.

3.3. Procesamiento por Lote en el desarrollo de sistemas bancarios

Con el objetivo de brindar como ejemplo el uso del procesamiento por lote en el desarrollo de tecnologías para sistemas bancarios, se ha seleccionado la empresa “Modhelus”.

3.3.1. Empresa “Modhelus”

MODHELUS nace en 1996 como una división de EMILIO BARREIRA & ASOCIADOS S.A., consultora que desde 1982 se ha especializado en el desarrollo de tecnologías para el negocio de las instituciones financieras.
Su creación surge de la necesidad de nuevas arquitecturas para resolver problemas que las arquitecturas tradicionales no son capaces de solucionar.
Aprovechando más de veinte años de experiencia en proyectos y desarrollos para la banca y los negocios financieros, los conocimientos adquiridos por la investigación, y el análisis de las tendencias tecnológicas, MODHELUS ha aportado avances significativos en la maduración e implementación de nuevas arquitecturas tecnológicas para el negocio financiero.

El conocimiento, la experiencia demostrada y la capacidad de innovación que plantean sus desarrollos han merecido el apoyo y soporte financiero de la Agencia Nacional de Promoción Científica y Tecnológica, a través de líneas específicas del Banco Interamericano de Desarrollo, lo cual les ha permitido posicionarse como líderes y pioneros en el desarrollo e implementación de estas nuevas arquitecturas en América Latina, como así también en construir y diseñar “una nueva generación de soluciones para el negocio financiero”.

3.3.2. Visión funcional de la arquitectura. Operatoria Batch en “Modhelus”

Para resolver la ejecución de operaciones fuera de línea que afectan a volúmenes masivos de datos y se estructuran en procesos que son corridos, ya sea manual o automáticamente desde un “programador de tareas”, se han reutilizado en todo lo posible los componentes y servicios de la operatoria online.

La arquitectura definida resuelve este rehúso debido a la abstracción sobre la que se montan los componentes básicos. Ellos desconocen de dónde vienen los datos que usan ni las condiciones de cuándo y dónde deben grabarse dichos datos, lo que permite implementar un tipo de proceso compuesto especial para los procesos Batch. Este mecanismo actúa en forma masiva, preparando el contexto a los componentes tal como ellos lo esperan, resolviendo la estrategia de iteración y manejo del lote de datos a procesar.

Una operación Batch toma objetos desde un cursor de la base de datos, armando el contexto con los objetos obtenidos de a uno e invocando a los componentes deseados. Esta estrategia permite reducir el acceso a la base de datos, pre-cargando las entidades según sean necesarias. Dispone de lógicas especiales, según las cuales se pueden efectuar operaciones bajo modo de iteraciones. Así es posible definir la necesidad de un commit de la base de datos para evitar que crezca demasiado el “rollback segment” que el motor de la base asocia a la transacción en curso.
Otro posible uso de estas lógicas especiales es la necesidad de dividir la tarea a efectuar en lotes por reglas de negocio, de modo de procesar todo un lote relacionado en forma conjunta. De esta manera se pueden ir cargando uno a uno datos asociados a ese lote que se procesa.

La arquitectura permite múltiples motores de ejecución Batch y la creación de operaciones Batch ad-hoc de manera de alcanzar el máximo rendimiento. No obstante, el núcleo de una operación Batch está compuesto por los mismos componentes básicos que resuelven la lógica de negocio. Cada sección de una operación Batch se configura como si fuera un proceso puntual.
Se han previsto dos alternativas para el caso que un proceso Batch no llegue a término satisfactoriamente.

- En la primera, los datos necesarios para retomar el proceso están incluidos entre las tablas del modelo de negocio.

- En la segunda alternativa, el proceso mismo es idempotente pudiéndose correr las veces que fuera necesario manteniendo información extrínseca al modelo. En este caso la operación Batch deberá recordar información para su retoma.

3.3.3. Diagrama estático de la arquitectura.

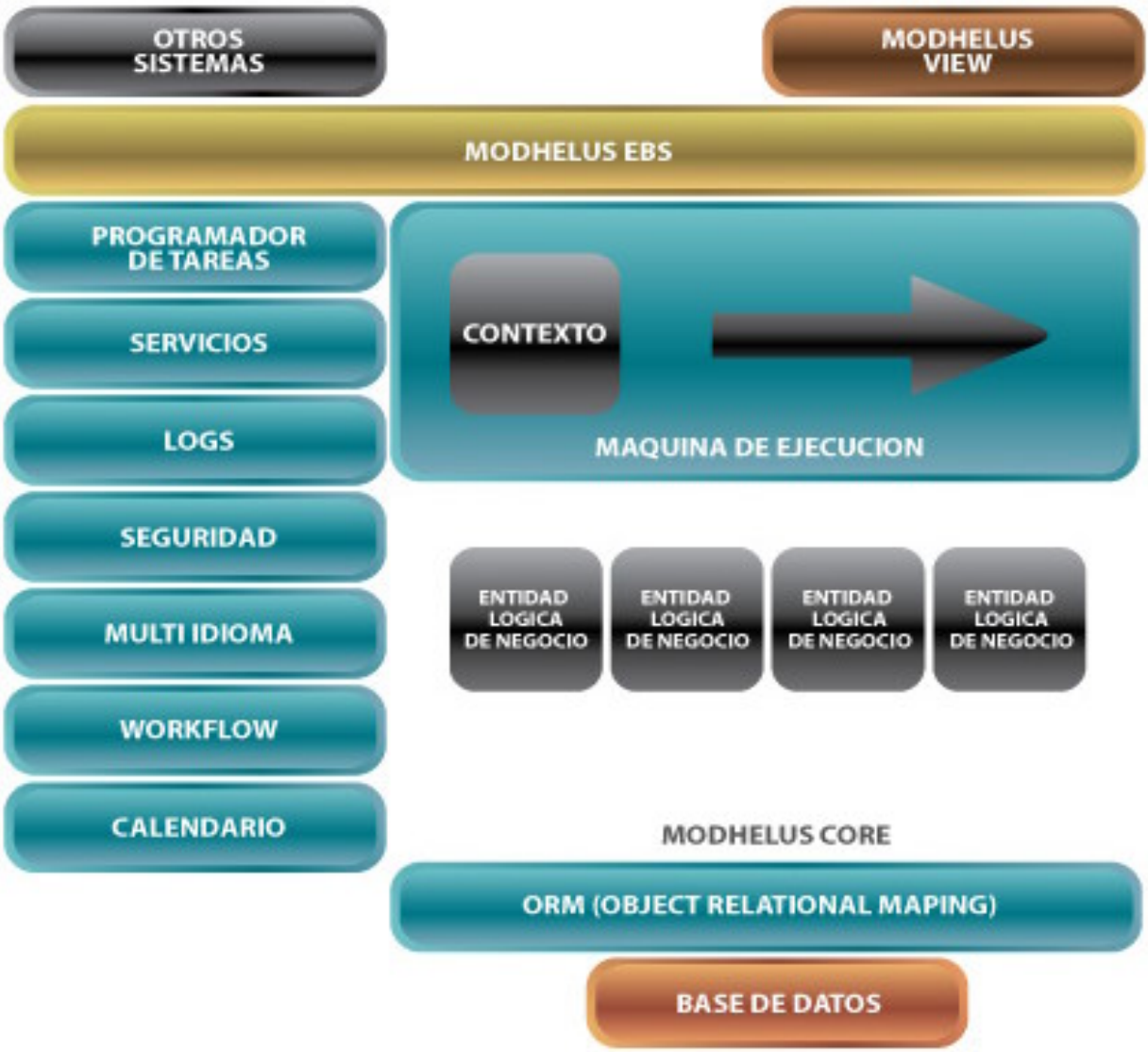

Ilustración 1: Diagrama estático de la arquitectura de "Modhelus"

4. Los Framework y el Procesamiento por Lote

4.1. Introducción

En la actualidad existen varias herramientas imprescindibles en el proceso de desarrollo de un software, entre las que se encuentran los Framework. "*Un Framework es un diseño reutilizable de todo o parte de un sistema de software descrito por varias jerarquías de herencia de clases (generalmente algunas abstractas), y por las colaboraciones que se establecen entre las instancias de estas clases."*[6]

Últimamente están siendo muy utilizados, pues su aplicación trae consigo grandes ventajas en el desarrollo de sistemas informáticos. Brindan una estructura perfectamente aplicable a la mayoría de los sistemas que se desarrollan actualmente en todo el mundo, pues proponen soluciones estándares apoyadas en la metodología de los lenguajes sobre los que se sustentan. Garantizan la simplificación de código en la mayoría de los casos, mediante librerías, clases hereditarias, y objetos; haciendo que estos gestionen internamente la estructura básica del sistema que se esté desarrollando.

Cada entorno de desarrollo cuenta con diferentes Framework, como por ejemplo: J2EE (Hibérnate, Spring, DOJO, Tapestry, WebWork), PHP (Symfony, Zoop, CodeIgniter, Zend), .NET (Entity, ASP, ADO, XML); aunque existen Framework que implementan distintas versiones para distintos lenguajes, como por ejemplo: Spring (J2EE, .NET).

En lo adelante este trabajo se centrará en un estudio más profundo de dos de los Framework mencionados, ya que estos califican como los más aplicables al Sistema Bancario que actualmente se está desarrollando, y finalmente concluyendo en uno como propuesta de solución:

- Hibérnate
- Spring

A continuación se presenta a modo de resumen el estudio realizado sobre estos Framework, y su manera de resolver la gestión de grandes cantidades de información en un corto intervalo de tiempo a través del procesamiento por lotes.

[6] **1. Cita.** R. E. Johnson y B. Foote. Designing reusable classes. Journal of Object-Oriented Programming, Capítulo 1(Tema 2): Páginas 22-35, 1988.

4.2. Framework Hibérnate

4.2.1. Introducción

Hibérnate nace a finales del 2001, y actualmente es la solución ORM (Object-Relational Mapping) más popular en el mundo Java.

Es un potente mapeador[7] objeto/relacional y servicio de consultas para Java, permite desarrollar clases persistentes a partir de clases comunes, incluyendo asociación, herencia, polimorfismo, composición y colecciones de objetos. El lenguaje de consultas de Hibérnate HQL (Hibérnate Query Language), diseñado como una mínima extensión orientada a objetos de SQL, proporciona un puente elegante entre los mundos objetual y relacional. Hibérnate también permite expresar consultas utilizando SQL nativo o consultas basadas en criterios. Soporta todos los sistemas gestores de bases de datos SQL y se integra sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web.

Uno de los posibles procesos de desarrollo consiste en, una vez tengamos el diseño de datos realizado, mapear este a ficheros XML siguiendo la DTD de mapeo de Hibérnate. Desde estos podremos generar el código de nuestros objetos persistentes en clases Java y también crear BBDD independientemente del entorno escogido.

Hibérnate se integra en cualquier tipo de aplicación justo por encima del contenedor de datos. Una posible configuración básica de Hibérnate es la siguiente:

Ilustración 2: Arquitectura Base

[7] **1. Sustantivo *m.***: Alguien o algo que mapea.

Nota: Se puede observar como Hibérnate utiliza la BBDD y la configuración de los datos para proporcionar servicios y objetos persistentes a la aplicación que se encuentre justo por arriba de él.

Entre otras cualidades importantes, Hibérnate le brinda una funcionalidad completamente configurable que le atribuye al sistema que se encuentre desarrollando, las herramientas necesarias para procesar grandes volúmenes de datos e información, a través de sus técnicas de procesamiento por lotes.
Y finalmente no se podría pasar por alto el hecho de que es software libre (open source), por lo que su utilización en el desarrollo de sistemas informáticos, no está sujeta a ninguna licencia, lo cual libera de cualquier compromiso y restricción si se decidiera optar por este Framework como propuesta de solución.

4.2.2. Características esenciales

De manera enumerativa se mencionan a continuación algunas de las características más importantes que posee este Framework, con el objetivo de describirlas brevemente.

- Persistencia transparente: Hibérnate puede operar proporcionando persistencia de una manera transparente para el desarrollador.
- Modelo de programación natural: Hibérnate soporta el paradigma de orientación a objetos de una manera natural (herencia, polimorfismo, composición y el Framework de colecciones de Java).
- Soporte para modelos de objetos con una granularidad muy fina: Permite una gran variedad de mapeos para colecciones y objetos dependientes.
- Sin necesidad de mejorar el código compilado (bytecode): No es necesaria la generación de código ni el procesamiento del bytecode en el proceso de compilación.
- Escalabilidad extrema: Hibérnate posee un alto rendimiento, tiene una caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa (lazy) de objetos y colecciones.

- Lenguaje de consultas HQL: Este lenguaje proporciona una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- Soporte para transacciones de aplicación: Hibérnate soporta transacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución) y gestiona la política optimistic locking automáticamente.
- Generación automática de claves primarias: Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos (secuencias, columnas autoincreméntales,...) así como generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.
- Soporte para procesamiento por lotes (Batch): Hibérnate soporta el procesamiento de grandes volúmenes de información en modo Batch (aquellas que no requieren la interacción con el usuario durante su ejecución).

4.2.3. Procesamiento por Lote en Hibérnate

El procesamiento por lotes es una fuerte característica de Hibérnate, particularmente cuando se quieren importar grandes lotes de datos de otro sistema. Si no se hace un buen uso de esta característica de Hibérnate, el rendimiento de la aplicación que se esté desarrollando podría disminuir considerablemente en el momento de grandes inserciones de información.

Existen dos enfoques para procesar información por lote en Hibérnate. Cada una de ellas será explicada a continuación:

➢ Un enfoque está relacionado con la clase Sesión (Session class). Para un mejor entendimiento en la explicación de este enfoque se usará el siguiente ejemplo:
Supongamos que hay una aplicación para una biblioteca que usa Hibérnate como capa ORM. Ahora usted quiere insertar 1 millón de libros en la biblioteca. Para mayor sencillez el modelo de dominio contiene dos clases, una es “libro”, y la segunda es “editor”.
El “editor” es contenido en la clase “libro”, y yo quiero habilitar la inserción y actualización en cascada, así que si inserto un libro con un editor que no está en el sistema, el editor también será guardado con el libro.

El código para insertar libros se muestra a continuación…

```
Session session =HibernateUtil.getSessionFactory().openSession();
session.beginTransaction();
for (int index = 0; index <> Book book = new Book();
{
 book.setAuthor("amer");
 book.setIsbn("34343");
 book.setName("Hibernate " + index);
 Publisher pub = new Publisher();
 pub.setName("Publisher " + index);
 book.setPublisher(pub);
 book.setPublishDate(new Date());
 session.save(book);
}
session.getTransaction().commit();
session.close();
```

Este código provoca mucha sobrecarga de memoria cada vez que este objeto es salvado, Hibérnate pone este objeto en una cache, la cual es llamada "Caché de sesión" o "Cache de primer nivel" y esto probablemente provoque el error de memoria “desbordamiento de pila” (Stack outofflow). Para evitar esto, se tiene que limpiar la sesión, pero se debe tener en cuenta cuando es recomendable limpiar la sesión, si después de una inserción, o, si después de un intervalo. Si se limpia la sesión, después de cada inserción, se reducirá dramáticamente el rendimiento de la aplicación, porque antes de llamar la operación de limpiar la sesión (Clear), se debe realizar la operación de poner al mismo nivel (Flush) todos los datos persistentes, y almacenarlos en la memoria de estado de objetos.

Para esta situación, se introduce un tamaño en el procesamiento por lote. Antes que nada se tendrá que añadir la propiedad de "Hibernate.jdbc.batch_size" en el archivo de hibernate.cfg.xml, con un valor que brinde el mejor rendimiento posible según el tamaño del lote de información a procesar, y de las características tecnológicas de la PC, en el ejemplo actual se eligió “50”. Usando esta propiedad, Hibérnate usa el jdbc para insertar 50 registros a la vez cuando se nivela (Flush) la sesión.

A continuación se muestra el código que se debe introducir para que se nivele y vacíe la sesión…

```
session.save(book); if (index % 50== 0)
{
 session.flush(); session.clear();
}
```

- La segunda manera de hacer el procesamiento por lote, es a través de la clase de "StatelessSession". La clase de “StatelessSession” difiere de la clase de “Session class”, en que no posee una cache para los objetos, no llama a interceptores, no salva ninguna persistencia de objetos, no permite poner los objetos en cascada, y transfiere directamente el objeto a la expresión de inserción del jdbc. Por lo tanto en otras palabras está más cerca del jdbc.

Con “StatelessSession”, se pueden salvar los objetos compuestos de manera separada, en el ejemplo que se está viendo, sería posible insertar los libros y los editores indistintamente.

El código con “StatelessSession” sería el siguiente…

```
StatelessSession session =
HibernateUtil.getSessionFacgtory().openStatelessSession();
session.beginTransaction();
for (int index = 0; index < style="font-style: italic; color: rgb(255,
153, 102);">
{
 Book book = new Book();
 book.setAuthor("amer");
 book.setIsbn("34343");
 book.setName("Hibernate " + index);
 Publisher pub = new Publisher();
 pub.setName("Publisher " + index);
 book.setPublisher(pub);
 book.setPublishDate(new Date());
 session.insert(pub);
 session.insert(book);
}
session.getTransaction().commit();
session.close();
```

4.2.4. Ejemplo

Es necesario insertar 200,000 entradas en una Base de Datos en Hibérnate. Para esto se necesita hacer los siguientes ajustes:

```
//set the JDBC batch size (it is fine somewhere between 20-50)
hibernate.jdbc.batch_size 30
```

```
//disable second-lavel cache
hibernate.cache.use_second_level_cache false
//and now do your job like this
Session S=SF.openSession();
//SF = SessionFactory object
Transaction T=S.beginTransaction();

for (int i=0;i<200000;i++)
{
 record r=new record(...);
 S.save(record);
 if(i % 30==0)
 {
  //30, same as the JDBC batch size
  //flush a batch and release memory
  session.flush();
  session.clear();
 }
}
//clean
T.commit();
S.close();
```

4.2.5. Conclusión de Hibérnate

Después de haberse empleado las dos maneras de procesamiento por lote explicadas anteriormente.

Se recomienda usar la primera:

- “StatelessSession” no provee ningún rendimiento durante la sesión ni tampoco cuando se tiene que salvar instancias compuestas.
- Incluso para un objeto plano, “StatelessSession” no proporciona mucho rendimiento durante la sesión. Solamente proporciona rendimiento cuando un objeto contiene uno o dos atributos.

4.3. Spring Framework

4.3.1. Introducción

Spring es un Framework de aplicaciones Java/J2EE, aunque también existe una versión para la plataforma .NET (Spring.net). Es un software de código abierto. La primera versión fue escrita por Rod Johnson, quien lo lanzó primero con la publicación de su libro *Expert One-on-One Java EE Design and Development*. El Framework fue lanzado inicialmente bajo Apache 2.0 en junio de 2003. El primer gran lanzamiento hito fue la versión 1.0, que apareció en marzo de 2004 y fue seguida por otros hitos en septiembre de 2004 y marzo de 2005.

A pesar de que Spring Framework no obliga a usar un modelo de programación en particular, se ha popularizado en la comunidad de programadores en Java al considerársele una alternativa, y sustituto del modelo de Enterprise JavaBean. Por su diseño el Framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria. Las características fundamentales de este Framework pueden emplearse en el desarrollo de cualquier aplicación hecha en Java.

4.3.2. Historia

Los primeros componentes de lo que se ha convertido en Spring Framework fueron escritos por Rod Johnson en el año 2000, mientras trabajaba como consultor independiente para sus clientes en la industria financiera en Londres. Mientras escribía el libro Expert One-on-One J2EE Design And Development (Programmer to Programmer), Rod amplió su código para sintetizar su visión acerca de cómo las aplicaciones que trabajan con varias partes de la plataforma J2EE podían llegar a ser más simples y más consistentes que aquellas que los desarrolladores y compañías estaban usando por aquel entonces.

En el año 2001 los modelos dominantes de programación para aplicaciones basadas en web eran ofrecidas por el API Java Servlet y los Enterprise JavaBeans, ambas especificaciones creadas por Sun Microsystems en colaboración con otros distribuidores y partes interesadas que disfrutaban

de gran popularidad en la comunidad Java. Las aplicaciones que no eran basadas en web, como las aplicaciones basadas en cliente o aplicaciones en Batch, podían ser escritas con base en herramientas y proyectos de código abierto o comercial que proveyeran las características requeridas para aquellos desarrollos.

Rod Johnson es reconocido por crear un Framework que está basado en las mejores prácticas aceptadas, y ello las hizo disponibles para todo tipo de aplicaciones, no sólo aquellas basadas en web. Estas ideas también estaban plasmadas en su libro y, tras la publicación, sus lectores le solicitaron que el código que acompañaba al libro fuera liberado bajo una licencia open source.

Se formó un pequeño equipo de desarrolladores que esperaba trabajar en extender el Framework y un proyecto fue creado en Sourceforge en febrero de 2003. Después de trabajar en su desarrollo durante más de un año lanzaron una primera versión (1.0) en marzo de 2004. Después de este lanzamiento Spring ganó mucha popularidad en la comunidad Java, debido en parte al uso de Javadoc y de una documentación de referencia por encima del promedio de un proyecto de código abierto.

Sin embargo, Spring Framework también fue duramente criticado en 2004 y sigue siendo el tema de acalorados debates. Al tiempo en que se daba su primer gran lanzamiento muchos desarrolladores y líderes de opinión vieron a Spring como un gran paso con respecto al modelo de programación tradicional; esto era especialmente cierto con respecto a Enterprise JavaBeans. Una de las metas de diseño de Spring Framework es su facilidad de integración con los estándares J2EE y herramientas comerciales existentes. Esto quita en parte la necesidad de definir sus características en un documento de especificación elaborado por un comité oficial y que podría ser criticado.

Spring Framework hizo que aquellas técnicas que resultaban desconocidas para la mayoría de programadores se volvieran populares en un periodo muy corto de tiempo. El ejemplo más notable es la inversión de control. En el año 2004, Spring disfrutó de unas altísimas tasas de adopción y al ofrecer su propio Framework de programación orientada a aspectos (aspect-oriented programming, AOP) consiguió hacer más popular su paradigma de programación en la comunidad Java.

En 2005 Spring superó las tasas de adopción del año anterior como resultado de nuevos lanzamientos y más características fueron añadidas. El foro de la comunidad formada alrededor de Spring Framework (The Spring Forum) que arrancó a finales de 2004 también ayudó a incrementar la popularidad del Framework y desde entonces ha crecido hasta llegar a ser la más importante fuente de información y ayuda para sus usuarios.

En el mismo año los desarrolladores del proyecto abrieron su propia compañía para ofrecer soporte comercial y establecieron una alianza con BEA. En diciembre de 2005 la primera conferencia de Spring fue realizada en Miami y reunió a 300 desarrolladores en el transcurso de tres días, seguida por una conferencia en Amberes en junio de 2006, donde se concentraron más de 400 personas.

4.3.3. Características esenciales

- Una potente gestión de configuración basada en JavaBeans, aplicando los principios de Inversión de Control (IoC).
  Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener singletons ni ficheros de configuración, una aproximación consistente y elegante. Esta factoría de beans puede ser usada en cualquier entorno, desde applets hasta contenedores J2EE. Estas definiciones de beans se realizan en lo que se llama el contexto de aplicación.
- Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción enchufables (pluggables), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel.
  Se incluyen estrategias genéricas para JTA y un único JDBC DataSource. En contraste con el JTA simple o EJB CMT, el soporte de transacciones de Spring no está atado a entornos J2EE.
- Una capa de abstracción JDBC que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores, y reduce considerablemente la cantidad de código necesario.
- Integración con Hibérnate, JDO e iBatis SQL Maps en términos de soporte a implementaciones DAO y estrategias con transacciones.
  Especial soporte a Hibérnate añadiendo convenientes características de IoC, y solucionando muchos de los comunes problemas de integración de Hibérnate. Todo ello cumpliendo con las transacciones genéricas de Spring y la jerarquía de excepciones DAO.

- Funcionalidad AOP, totalmente integrada en la gestión de configuración de Spring. Se puede aplicar AOP a cualquier objeto gestionado por Spring, añadiendo aspectos como gestión de transacciones declarativa.
  Con Spring se puede tener gestión de transacciones declarativa sin EJB, incluso sin JTA, si se utiliza una única base de datos en un contenedor web sin soporte JTA.
- Un Framework MVC (Model-View-Controller), construido sobre el núcleo de Spring. Este Framework es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser JSP, Velocity, Tiles, iText o POI.
  De cualquier manera una capa modelo realizada con Spring puede ser fácilmente utilizada con una capa web basada en cualquier otro Framework MVC, como Struts, WebWork o Tapestry.

Todas estas funcionalidades pueden emplearse en cualquier servidor J2EE, y la mayoría de ellas ni siquiera lo requieren. El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables, no atados a servicios J2EE específicos. La arquitectura en capas de Spring ofrece una gran flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en JavaBeans sin utilizar el Framework MVC o el soporte AOP.

4.3.4. Procesamiento por Lote en Spring

Spring Framework, brinda una herramienta completamente dedicada al procesamiento por Lote, muy conocida dentro del mundo Java con el nombre de Spring Batch.
Este Framework ha revolucionado el concepto de “Procesamiento por Lote”, pues además de aplicar los principios generales de la mayoría de las aplicaciones que desarrollan esta técnica, como son la lectura y escritura de datos, en lotes de grandes cantidades de información; Spring Batch añade las herramientas necesarias para transformar, modificar, o procesar estos lotes de información permitiendo la interacción de los datos contenidos en ellos con otros Framework dentro de la misma aplicación que se esté desarrollando, así como con las bases de datos, y con aplicaciones externas.

Inicialmente este Framework brinda una serie de principios básicos necesarios a tener en cuenta durante la construcción de las arquitecturas Batch:

- Las arquitecturas Batch influyen directamente sobre las arquitecturas en Línea, y viceversa… Se debe diseñar con ambas arquitecturas y ambientes en mente, haciendo uso de componentes básicos comunes, siempre que sea posible.
- Es recomendable simplificar tanto como sea posible, y evitar construir estructuras lógicas complicadas en aplicaciones Batch sencillas.
- Se debe procesar los datos cerca de donde residen físicamente, o viceversa.
- Minimizar el uso de recursos del sistema, especialmente la E/S, llevando a cabo la menor cantidad de operaciones posibles en la memoria interna.
- Examine la aplicación de E/S (analice sentencias SQL) para asegurar que es evitada la E/S innecesaria. En especial para evitar las siguientes cuatro fallas comunes:
  - Leer los datos para cada transacción, cuando los datos pueden ser leídos una sola vez y guardados en el almacenamiento activo.
  - Releer datos para una transacción, cuando los datos ya fueron leídos anteriormente para esa misma transacción.
  - Causar tablas innecesarias o escaneo indexado.
  - No especificar valores de llave en clausulas WHERE de una declaración SQL.
- No es recomendable repetir las mismas acciones dos veces en un mismo proceso Batch.
- Realice la asignación de memoria al principio del procesamiento Batch, para evitar la reasignación durante el proceso.
- Siempre se debe asumir lo peor con respecto a la integridad de los datos. Inserte chequeos adecuados y validaciones para mantener la integridad de los datos.
- Implemente listas de chequeos para la validación interna siempre que sea posible.
- Planee y ejecute pruebas de tensión tan pronto como pueda en los ambientes de producción, en entornos con volúmenes de datos reales.
- En grandes sistemas Batch los respaldos de datos pueden variar constantemente, principalmente si el sistema funciona en línea, 24 horas al día los 7 días a la semana. Las copias de seguridad de la Base de Datos deben ser tomadas típicamente cuidando el diseño en línea. Si el sistema depende de archivos planos, los procedimientos de

seguridad de archivos deben ser documentados en un lugar, para evaluarlos con regularidad.

Spring Batch, como la mayoría de los componentes de Spring está basado en objetos POJO[8] (Plain Old Java Object), lo cual significa que hace uso de clases simples sin dependencia específica de ningún Framework, esto le ofrece facilidades al programador durante la implementación, brindándole funcionalidades como:

- Logging.
- Estadísticas.
- Manejo de transacciones.
- Suspensión y reanudación de Jobs.
- Manejo de recursos

Este componente además cuenta con diferentes estrategias para resolver problemas de procesamiento por lote, las cuales se pueden utilizar de manera independiente.

Las opciones típicas de procesamiento son:

- Procesamiento normal en una ventana de lote estando off-line.
- Procesamiento concurrente lote/on-line.
- Procesamiento paralelo de muchas y diferentes tareas de lote al mismo tiempo.
- Particionamiento (ej. Procesamiento de muchas instancias de la misma tarea de lote al mismo tiempo).
- Una combinación de estas.

[8] POJO (Plain Old Java Object) es una sigla creada por Martin Fowler, Rebecca Parsons y Josh MacKenzie en septiembre del 2000, y utilizada por programadores java para enfatizar el uso de clases simples y que no dependen de un Framework en especial. Este acrónimo surge como una reacción en el mundo Java a los Frameworks cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando. En particular surge en oposición al modelo planteado por los estándares EJB anteriores al 3.0, en los que los "Enterprise JavaBeans" debían implementar interfaces especiales. POJO es una nueva palabra para designar algo viejo. No existe en Java una nueva tecnología con ese nombre, sino que el nombre existe en el marco de una revalorización de la programación "simplemente orientada a objetos". Esta revalorización tiene que ver también con el éxito de lenguajes orientados a objetos más puros y sencillos, que empezaron a tomar parte del mercado al que Java apunta, como Ruby y Python.

El orden en la lista anterior refleja la complejidad de implementación, el procesamiento en una ventana de lote en la más fácil y el particionamiento es la más compleja de implementar.
Algunas o la totalidad de estas opciones pueden ser soportadas por un organizador comercial (Schedule).
En la siguiente sección esas opciones de procesamiento serán discutidas en mayor detalle. Es importante notar que la estrategia de Acometer y Proteger (commit and locking) adoptada por los procesos de lotes dependerá del tipo de procesamiento realizado, y como regla general y estrategia de protección on-line deberán también seguir los mismos principios. Por esta razón, la arquitectura del lote no puede ser simplemente dejada a un lado cuando se diseña una arquitectura general.
La estrategia de protección puede usar las protecciones normales de las bases de datos o en adición puede ser implementado un servicio de protección en la arquitectura. El servicio de protección debe seguir la protección propia de la base de datos (por ejemplo almacenando la información necesaria en una tabla dedicada) y otorgar o denegar permisos a los programas que solicitan operaciones en la base de datos. La lógica de reintento debe ser implementada usando esta arquitectura para evitar abortar una tarea de lote en caso de una situación de protección (lock situation).

1. **Procesamiento normal en una ventana de lote:** para procesos simples de lote ejecutándose en ventanas separadas, donde los datos que se actualizan no son requeridos por usuario on-line u otro proceso de lote, la concurrencia no es un problema y un “Acometer” (commit) puede ser hecho al final del proceso.
En la mayoría de los casos una aproximación más robusta es más apropiada. Una cosa a tener en mente es que cada sistema de lote tiene una tendencia a crecer según pasa el tiempo, en dos sentidos: complejidad y volumen de datos que deben manejar. Si no se coloca una estrategia de protección (locking estrategy) y el sistema aún recae en un solo punto de “Acometer” (commit point), modificar el programa de lote puede ser doloroso. Por esto, aun con el más simple de los sistemas de lote, considere la necesidad de incluir lógica para las funciones de reinicio-recuperación así como la información concerniente a casos más complejos a continuación dada.

2. **Procesamiento concurrente de lote/on-line:** el procesamiento de datos en aplicaciones de lotes puede ser actualizado simultáneamente por usuarios on-line, no se debe proteger (lock) cualquier dato que pueda ser requerido por usuarios on-line por más de pocos segundos. También

las actualizaciones deben ser realizadas en la base de datos al final de cada nueva transacción. Esto minimiza la porción de datos que queda inaccesible a otros procesos y el tiempo en que permanece en este estado.

Otra opción para minimizar la protección física es contar con una protección lógica implementada a nivel de fila usando un Modelo de Protección Optimista (Optimistic Locking Pattern) o un Modelo de Protección Pesimista (Pesimistic Locking Pattern).

- La Protección Optimista asume una baja probabilidad de conflictos en los registros. Normalmente indica el uso de columna para la fecha en cada tabla de la base de datos usada concurrentemente por los procesos de lote y on-line. Cuando una aplicación obtiene una fila para su procesamiento, también obtiene la fecha. Como la aplicación trata de actualizar la fila procesada, la actualización usa la fecha original en la cláusula WHERE. Si la fecha coincide, los datos serán actualizados satisfactoriamente. Si la fecha no coincide, esto indica que otra aplicación accede a la misma fila entre el momento de la obtención y el intento de actualización y por tanto la actualización no puede ser realizada.
- La Protección Pesimista es cualquier estrategia de protección que asuma que: existe una alta probabilidad de conflicto en los registros y entonces se necesita una protección lógica o física en el momento de la recuperación. Un tipo de protección pesimista lógica usa una columna de protección dedicada en la tabla de la base de datos. Cuando la aplicación obtiene la fila para actualizarla, pone una bandera en la columna. Con la bandera puesta, otras aplicaciones que intenten obtener la misma fila lógicamente fallarán. Cuando la aplicación que puso la bandera actualice la fila, también quitará la bandera, habilitándola para ser obtenida por otras aplicaciones. Por favor notar que la integridad de los datos debe ser mantenida entre la obtención integral y la colocación de la bandera, por ejemplo usando las protecciones de SQL (ej. SELECT FOR UPDATE). Nótese también que este método sufre el mismo problema que la protección física, solo que es un poco más fácil lidiar con la construcción de un mecanismo tiempo que garantice la liberación de la protección si el usuario sale a almorzar mientras el registro está protegido.

Estos modelos no son necesariamente apropiados para procesamiento de lotes, pero pueden ser usados para el procesamiento concurrente de lotes/on-line (ej. En casos donde la base de datos no soporte una protección a nivel de fila). Como regla general, la protección optimista es más adecuada para aplicaciones on-line, mientras que la protección pesimista es más propicia para

aplicaciones de lote. Siempre que sea usada la protección lógica, el mismo esquema debe ser usado para todas las aplicaciones que acceden a entidades de datos protegidas de manera lógica. Nótese que ambas soluciones lidian con la protección a una sola fila. A menudo necesitamos proteger a un grupo de registros relacionados lógicamente. Usando cierres físicos, se debe manejar esto con sumo cuidado para evitar “cierres fatales” potenciales. Con cierres lógicos, normalmente es mejor construir un manejador de cierres (lock manager) que comprenda los grupos lógicos de registros que se quieren cerrar o proteger y que garantice que dichos cierres sean coherentes y no fatales.

3. **Procesamiento en Paralelo:** El procesamiento en paralelo permite ejecutar múltiples tareas de lote en paralelo para minimizar el tiempo total de procesamiento. Esto no es un problema mientras las tareas no estén compartiendo los mismos archivos, bases de datos o espacios de índice. Si lo hacen, este servicio deberá ser implementado usando datos particionados. Otra opción es construir un módulo de arquitectura para mantener la interdependencia usando una tabla de control. Una tabla de control deberá contener una fila por cada recurso compartido y si está siendo usado por alguna aplicación o no. La arquitectura del lote o de la aplicación en paralelo deberá entonces obtener de la tabla la información que le permita determinar si puede acceder al recurso que necesita o no.
Si el acceso a los datos no es un problema, el procesamiento en paralelo puede ser implementando a través del uso de hilos adicionales en paralelo. En el ambiente de “mainframe” (supercomputadoras), las clases de tareas en paralelo han sido usadas tradicionalmente, para asegurar el tiempo de CPU adecuado para cada proceso. De cualquier manera, la solución tiene que ser lo suficientemente robusta para garantizar porciones de tiempo para todos los procesos en ejecución.
Otro tema importante en el procesamiento en paralelo incluye el balance de carga y la disponibilidad de los recursos generales del sistema tales como: archivos, bases de datos, buffers, etc. Nótese también que la tabla de control en si misma pude convertirse fácilmente en un recurso crítico del sistema.

4. **Particionamiento:** El uso del particionamiento permite la ejecución concurrente de múltiples versiones de aplicaciones de lote muy extensas. El propósito de esto es reducir el tiempo requerido para procesar tareas de lotes extensas. Los procesos que pueden ser particionados satisfactoriamente son aquellos donde el archivo de entrada pueda ser separado (split) y/o las

tablas de la base de datos principal puedan ser particionadas para permitir que la aplicación corra con diferentes juegos de datos.

En adición, los procesos que son particionados deben diseñarse para tratar solamente con los datos que le son asignados. Una arquitectura de particionamiento debe estar estrechamente vinculada al diseño de la base de datos y a la estrategia de particionamiento de la misma. Por favor tener en cuenta que, el particionamiento de la base de datos no involucra necesariamente una segmentación física de la base de datos, a pesar de que en la mayoría de los casos es recomendable.

En la siguiente figura se muestra una propuesta de particionamiento:

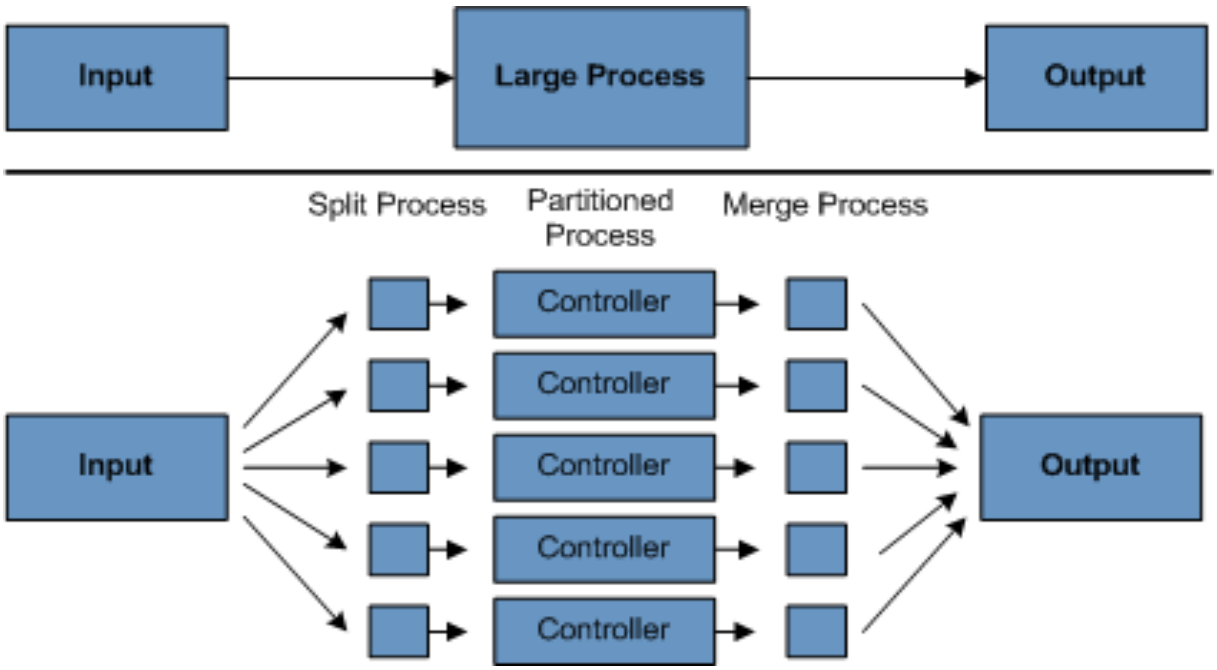

Ilustración 3: Propuesta de Particionamiento

La arquitectura debe ser lo suficientemente flexible para permitir la configuración del número de particiones de manera dinámica. Deben ser consideradas las opciones de configuración automática y vía usuario. La configuración automática deberá basarse en parámetros como el tamaño del archivo de entrada y/o el número de registros de entrada.

4.1 **Propuestas de Particionamiento:**

La siguiente lista muestra algunos de las posibles propuestas de particionamiento. La selección de una propuesta debe realizarse sobre el análisis caso-a-caso.

1. *Segmentado Predeterminado y Uniforme del Juego de Registros*

Esto implica la segmentación del juego de registros de entrada en un número uniforme de porciones (ej. 10, donde cada porción tendrá exactamente 1/10 del juego de datos.). Cada porción es entonces procesada por una instancia de la aplicación lote/extracción.

Para el uso de esta propuesta, se necesitará de procesamiento para la división del juego de registros. El resultado de esta división serán números de límites de posición inferior y superior, que en cada caso podrán ser utilizados como entrada de la aplicación de lote/extracción con el fin de restringirla a su porción solamente.

El pre-procesamiento puede ser muy costoso si tiene que calcular y determinar los límites de cada porción del juego de registros.

2. *Segmentado por Campo Llave*

Esto implica la división del juego de registros de entrada por un campo llave como puede ser el código de localidad, y a la asignación de los datos de cada llave a una instancia de lote. Para lograr esto, los valores del campo pueden ser:

2.1 Asignados a una instancia por una tabla de partición (ver abajo para más detalles)

2.2 Asignados a una instancia por una porción del valor (ej. Valores 1000-1999, etc.)

Bajo la opción 1, la adición de valores nuevos indicará una reconfiguración manual de la aplicación de lote/extracción para asegurar que el nuevo valor sea agregado a una instancia en particular.

Bajo la opción 2, esto asegurará que los valores son cubiertos por una instancia de la tarea de lote. Sin embargo, el número de valores procesados por una instancia es dependiente de la distribución de los valores del campo (ej. Puede haber un gran número de posiciones en el rango 0000-0009, y un pequeño número en el rango 1000-1999). Bajo esta opción, los rangos de datos deben ser diseñados con el particionamiento en mente.

Bajo ambas opciones, la distribución uniforme óptima de los registros a las instancias de lote no puede ser realizada. No hay configuración dinámica del número de instancias de lote usadas.

3. *Segmentado por Vistas*

Esta propuesta es básicamente “segmentado por campo llave”, pero a nivel de base de datos. Implica la segmentación de los juegos de registros en vistas. Estas vistas serán usadas por cada

instancia de la aplicación de lote durante su procesamiento. El segmentado será realizado agrupando los datos.

Con esta opción, cada instancia de la aplicación será configurada para incidir sobre una vista en particular (en vez de la tabla maestra). También, con la adición de nuevos valores, este nuevo grupo de datos será incluido en una vista. No existe capacidad de configuración dinámica, un cambio en el número de instancias resultara en un cambio en el número de vistas.

4. *Adición de un Indicador de Procesamiento*

Esto involucra la adición de una nueva columna dentro de la tabla de entrada, que actuará como un indicador. Como una acción de pre-procesamiento, todos los indicadores deberán ser marcados a “no-procesado”. Durante la etapa de adquisición de registros de la aplicación de lote, los registros son leídos bajo la condición de que estén macados “no-procesado”, y una vez que se leen (con cierre), son marcados “procesando”. Cuando ese registro es completado, el indicador es actualizado a “completado” o “error”. Muchas instancias de una aplicación de lote pueden ser iniciadas sin cambios, ya que el campo adicional asegura que un registro sea procesado solo una vez.

Con esta opción, E/S en la tabla aumenta dinámicamente. En caso de una aplicación de actualización, este impacto es reducido, ya que una escritura ocurrirá de cualquier manera.

Extracción de la Tabla a un Archivo Plano

Esto involucra la extracción de la tabla a un archivo. Este archivo puede ser segmentado en múltiples segmentos y ser usado como entrada de las instancias de la aplicación.

Con esta opción, el costo de operación adicional por la extracción de la tabla a un archivo y la segmentación del mismo, puede contrarrestar el efecto de multi-particionamiento. La configuración dinámica puede ser alcanzada modificando en código de segmentación.

5. *Uso de un Campo de Hash*

Este esquema involucra la adición de un campo de hash (llave/índice) a las tablas de la base de datos usado para recuperar el registro controlador. Este campo de hash tendrá un indicador para determinar cual instancia de la aplicación procesará una fila en particular. Por ejemplo, si tenemos tres instancias para ser iniciadas, entonces un indicador “A” marcará a la columna que procesará la instancia 1, un marcador “B” a la que procesará la instancia 2 y otro “C” para la asignada a la instancia 3.

El procedimiento usado para obtener los registros deberá tener una cláusula “WHERE” adicional para seleccionar todas las filas marcadas por un indicador en particular. La inserción en las tablas involucra la adición del campo marcador, el cual pudiera ser llenado por defecto a una de las instancias (ej. “A”).

Una aplicación de lote sencilla puede ser utilizada para actualizar los indicadores distribuyendo la carga entre diferentes instancias. Cuando un número lo suficientemente grande de filas ha sido agregado, este lote puede ser ejecutado (en cualquier momento, excepto en la ventana de lote) para redistribuir los indicadores y acomodarlos al nuevo número de instancias.

4.3.5. Ejemplo

```
public class FileDeletingTasklet implements Tasklet, InitializingBean
{
 private Resource directory;

 public ExitStatus execute() throws Exception
 {
  File dir = directory.getFile();
  Assert.state(dir.isDirectory());
  File[] files = dir.listFiles();
  for (int i = 0; i < files.length; i++)
  {
   boolean deleted = files[i].delete();
   if (!deleted)
   {
    throw new UnexpectedJobExecutionException("Could not delete file "
    + files[i].getPath());
   }
  }
  return ExitStatus.FINISHED;
 }

 public void setDirectoryResource(Resource directory)
 {
  this.directory = directory;
 }
 public void afterPropertiesSet() throws Exception
 {
  Assert.notNull(directory, "directory must be set");
 }
}
```

```
.....................................................................
<job id="taskletJob">
 <step id="deleteFilesInDir" tasklet="fileDeletingTasklet"/>
</job>

<bean id="fileDeletingTasklet"
    class="org.springframework.batch.sample.tasklet.FileDeletingTasklet">
  <property name="directoryResource">
   <bean id="directory"
       class="org.springframework.core.io.FileSystemResource">
    <constructor-arg value="target/test-outputs/test-dir" />
   </bean>
  </property>
</bean>
```

4.3.6. Conclusión de Spring Batch

Apoyado sobre una solida estructura compuesta por principios básicos, funcionalidades implementadas, estrategias definidas, y un potente lenguaje como lo es Java; Spring Batch facilita el procesamiento de grandes cantidades de información, atribuyéndole una gran robustez a las aplicaciones resultantes.

5. Las Bases de Datos y el Procesamiento por Lote

5.1. Introducción

El desarrollo progresivo de la Informática, así como la creciente y cada vez más exigente demanda de las aplicaciones que gestionan los numerosos datos del mundo informatizado en el que vivimos, han dado lugar al surgimiento de nuevos métodos de almacenamiento, con el objetivo de salva-guardar el caudal más preciado que poseen las sociedades en la actualidad, como lo es la información. Entre estos nuevos sistemas de almacenamiento, se encuentra la creación de las Bases de Datos (BD). *"Una Base de Datos es un conjunto de datos interrelacionados entre sí, almacenados con carácter más o menos permanente en la computadora. O sea, que una BD puede considerarse una colección de datos variables en el tiempo."*[9]

Las bases de datos y sus diversos modelos, presentan en la actualidad un rango de soluciones a la cuestión del almacenamiento de la información, y sus usos más comunes se encuentran en las operaciones de empresas e instituciones públicas, así como también en entornos científicos, educativos y de investigación.

El software que permite la utilización y/o la actualización de los datos almacenados en una (o varias) base(s) de datos por uno o varios usuarios desde diferentes puntos de vista y a la vez, se denomina Sistema de Gestión de Bases de Datos (SGBD). "Es importante diferenciar los términos BD y SGBD."

El objetivo fundamental de un SGBD consiste en suministrar al usuario las herramientas que le permitan manipular, en términos abstractos, los datos, o sea, de forma que no le sea necesario conocer el modo de almacenamiento de los datos en la computadora, ni el método de acceso empleado. Los programas de aplicación operan sobre los datos almacenados en la base utilizando las facilidades que brindan los SGBD, los que, en la mayoría de los casos, poseen lenguajes especiales de manipulación de la información que facilitan el trabajo de los usuarios.

[9] **1. Cita.** Lic. Rosa María Mato García. Diseño de Bases de Datos, Tema 1: Pagina 2, 1999.

Los SGBD y el Procesamiento por Lote están estrechamente vinculados, esto lo demuestran aplicaciones como Oracle y Microsoft SQL Server; las cuales hacen uso de esta técnica para facilitar las respuestas a consultas demasiado grandes que demandan la escritura o lectura de grandes bloques de información ubicados incluso en diferentes tablas.
Para un mejor entendimiento de este vinculo entre los SGBD y el PpL, se expone a continuación el estudio realizado.

5.2. Procesamiento por lotes en Microsoft SQL Server 2005 (Analysis Services)

Microsoft SQL Server 2005 Analysis Services (SSAS), puede procesar objetos por lotes. Al utilizar el procesamiento por lotes, puede seleccionar los objetos que se van a procesar y controlar el orden de procesamiento. Además, un lote se puede ejecutar como una serie de trabajos independientes o como una transacción en la que un error en un proceso causa la reversión del lote completo.

Puede ejecutar el procesamiento por lotes utilizando uno de estos métodos:

- Explorador de objetos en SQL Server Management Studio. Con este método, puede seleccionar objetos similares, como un conjunto de dimensiones o un conjunto de particiones, para el procesamiento por lotes. Para procesar otros objetos, puede seleccionar la opción Procesar objetos afectados para procesar las particiones afectadas por el procesamiento de dimensiones.
- Explorador de soluciones en Business Intelligence Development Studio. Este método proporciona la misma funcionalidad que el Explorador de objetos en Management Studio. Antes de procesar los objetos en BI Development Studio, se debe implementar el proyecto que contiene los objetos.
- Una secuencia de comandos XMLA, con la ventana Consulta XMLA en Management Studio o como una tarea programada. Puede crear y ejecutar una secuencia de comandos XMLA con Management Studio, como se describe en el siguiente procedimiento.

Los siguientes procedimientos muestran los pasos para procesar completamente dimensiones y particiones. El procesamiento por lotes también puede incluir otras opciones de procesamiento,

como el procesamiento incremental. Para que estos procedimientos funcionen correctamente, debe utilizar una base de datos de SQL Server 2005 Analysis Services (SSAS) existente que contenga al menos dos dimensiones y una partición. Se recomienda que ejecute estos procedimientos en un entorno de prueba, no en sistemas de producción.

Para crear y ejecutar un proceso por lotes con el Explorador de objetos en SQL Server Management Studio:

1. Abra Management Studio, busque una base de datos de Analysis Services y expanda el contenedor de base de datos.
2. Haga clic en la carpeta “Dimensiones” y, a continuación, en la ficha “Resumen”.
3. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la ventana “Resumen”.
4. Haga clic con el botón secundario en las dimensiones seleccionadas y active “Proceso”.
5. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la lista de objetos.
6. Haga clic con el botón secundario en las dimensiones seleccionadas y active “Procesar completo”.
7. Para personalizar el trabajo de proceso por lotes, haga clic en “Cambiar configuración”.
8. En “Opciones de procesamiento”, marque la siguiente configuración:
   La opción “Orden de procesamiento” establecida en “Secuenciales” y la opción “Modo de transacción” establecida en “Una transacción”.
   La opción “Opción de tabla de reescritura” establecida en “Utilizar existente”.
   En “Objetos afectados”, active la casilla de verificación “Procesar objetos afectados”.
9. Haga clic en la ficha “Errores de clave de dimensión”. Verifique que la opción “Utilizar la configuración de error predeterminada” esté activada.
10. Haga clic en “Aceptar” para cerrar la pantalla “Cambiar configuración”.
11. Haga clic en “Aceptar” en la pantalla “Procesar objetos” para iniciar el trabajo de procesamiento.
12. Cuando en el cuadro “Estado” se muestre “Proceso finalizado correctamente”, haga clic en “Cerrar.”

Para crear y ejecutar un proceso por lotes con el Explorador de soluciones en Business Intelligence Management Studio:

1. Abra BI Development Studio.
2. Abra un proyecto que se haya implementado.
3. En el “Explorador de soluciones”, debajo del proyecto implementado, expanda la carpeta “Dimensiones”.
4. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la carpeta “Dimensiones”.
5. Haga clic con el botón secundario en las dimensiones seleccionadas y, a continuación, haga clic en “Procesar”.
6. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la lista de objetos.
7. Haga clic con el botón secundario en las dimensiones seleccionadas y active “Procesar completo”.
8. Para personalizar el trabajo de proceso por lotes, haga clic en “Cambiar configuración”.
9. En “Opciones de procesamiento”, marque la siguiente configuración:
   La opción “Orden de procesamiento” establecida en “Secuenciales” y la opción “Modo de transacción” establecida en “Una transacción”.
   La opción “Opción de tabla de reescritura” establecida en “Utilizar existente”.
   En “Objetos afectados”, active la casilla de verificación “Procesar objetos afectados”.
10. Haga clic en la ficha “Errores de clave de dimensión”. Verifique que la opción “Utilizar la configuración de error predeterminada” esté activada.
11. Haga clic en “Aceptar” para cerrar la pantalla “Cambiar configuración”.
12. Haga clic en “Ejecutar” en la pantalla “Procesar objetos” para iniciar el trabajo de procesamiento.
13. Cuando en el cuadro “Estado” muestre “Proceso finalizado correctamente”, haga clic en “Cerrar”.
14. Haga clic en “Cerrar” en la pantalla “Procesar objetos”.

Para crear y ejecutar una secuencia de comandos XMLA con el Explorador de objetos en SQL Server Management Studio:

1. Puede escribir manualmente una secuencia de comandos XMLA para trabajar con Analysis Services con cualquier editor de texto, como el Bloc de notas. Sin embargo, puede utilizar Analysis Services para crear una secuencia de comandos XMLA en Management Studio que se pueda ejecutar en la ventana Consulta XMLA en cualquier equipo SQL Server 2005

Analysis Services (SSAS), o bien dentro de una tarea que se pueda programar. Este procedimiento muestra cómo crear y ejecutar una secuencia de comandos XMLA utilizando SQL Server Management Studio. Para obtener más información acerca de las tareas programadas, vea Programar tareas administrativas con el Agente SQL Server.

2. Abra Management Studio, busque una base de datos de Analysis Services y expanda el contenedor de base de datos.
3. Haga clic en la carpeta “Dimensiones” y, a continuación, en la ficha “Resumen”.
4. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la ventana “Resumen”.
5. Haga clic con el botón secundario en las dimensiones seleccionadas y active “Proceso”.
6. Manteniendo presionada la tecla Ctrl, haga clic en todas las dimensiones que aparecen en la lista de objetos.
7. Haga clic con el botón secundario en las dimensiones seleccionadas y active “Procesar completo”.
8. Para personalizar el trabajo de proceso por lotes, haga clic en “Cambiar configuración”.
9. En “Opciones de procesamiento”, marque la siguiente configuración:
   La opción “Orden de procesamiento” establecida en “Secuenciales” y la opción “Modo de transacción” establecida en “Una transacción”.
   La opción “Opción de tabla de reescritura” establecida en “Utilizar existente”.
   En “Objetos afectados”, active la casilla de verificación “Procesar objetos afectados”.
10. Haga clic en la ficha “Errores de clave de dimensión”. Verifique que la opción “Utilizar la configuración de error predeterminada” esté activada.
11. Haga clic en “Aceptar” para cerrar la pantalla “Cambiar configuración”.
12. En la pantalla “Procesar objetos”, haga clic en “Secuencia de comandos”. Este paso genera una secuencia de comandos XMLA y abre una ventana “Consulta XMLA” en la que se puede ejecutar la secuencia de comandos XMLA.
13. En la pantalla “Procesar objetos”, haga clic en “Cancelar” para cerrar la pantalla sin ejecutar el trabajo de procesamiento.
14. Cambie a la ventana “Consulta XMLA” y, a continuación, haga clic en “Ejecutar” para ejecutar la secuencia de comandos.

5.3. Ejemplo

Algunas aplicaciones requieren que se realicen en los datos operaciones de procesamiento intensivo por lotes. En muchos casos, estas operaciones por lotes no se pueden realizar en el servidor de procesamiento de transacciones en línea, porque la sobrecarga de procesamiento interfiere con otras operaciones del servidor. En este caso es necesario realizar el procesamiento por lotes en un servidor independiente. En algunos casos, el procesamiento por lotes simplemente se descarga; en otros, los resultados del proceso por lotes se propagan de nuevo al servidor de procesamiento en línea.
En el siguiente diagrama se muestra una situación típica con datos que se replican en un servidor de procesamiento por lotes:

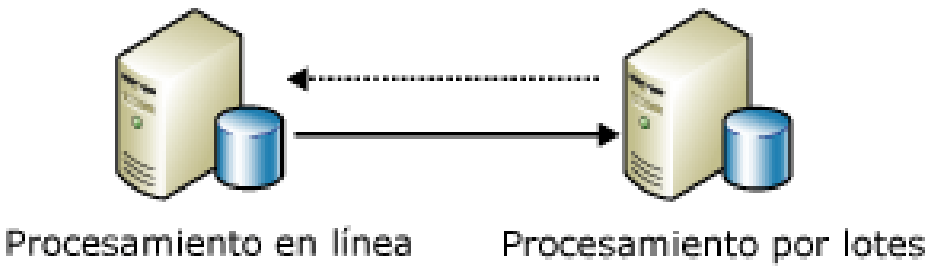

Ilustración 4: Replica de un Servidor de Procesamiento por Lotes

Ejemplo de Adventure Works Cycles:

Adventure Works Cycles es una compañía ficticia que se utiliza para mostrar situaciones y conceptos de bases de datos.
Adventure Works Cycles utiliza al procesamiento por lotes para comprobar si hay fraudes con tarjeta de crédito en su sitio Web. Los datos recopilados de las transacciones en el sitio Web se replican desde el servidor Microsoft SQL Server que da servicio al sitio Web en un servidor SQL Server independiente que se utiliza para una serie de aplicaciones de Adventure Works Cycles.
En el servidor de procesamiento por lotes, los datos se comprueban en busca de patrones de fraude con tarjetas de crédito. Aunque la detección del fraude produce una pequeña cantidad de datos (datos de actualización en un pequeño número de columnas si una cuenta muestra actividad sospechosa), las comprobaciones realizan una actividad intensiva de cálculo y requieren considerables recursos de servidor. Una vez que se ejecuta el proceso por lotes, se envía de

nuevo al servidor OLTP del sitio Web una pequeña cantidad de datos que indican las cuentas que muestran posibles indicios de fraude.

Requisitos habituales en este escenario:

- Las aplicaciones de procesamiento por lotes suelen tener los siguientes requisitos, que una solución de réplica adecuada debe satisfacer:
- El sistema debe mantener la coherencia transaccional.
- El sistema debe tener una latencia baja: las actualizaciones en el servidor de procesamiento por lotes deben llegar al servidor de procesamiento por lotes con rapidez.
- El sistema debe tener un rendimiento alto: debe controlar la réplica de un gran número de transacciones.
- El proceso de réplica debe producir una sobrecarga mínima en el servidor de procesamiento en línea.
- Los cambios de datos pueden fluir en ambas direcciones: los resultados del procesamiento por lotes pueden propagarse de nuevo al servidor de procesamiento en línea.
- Los datos requeridos en el servidor de procesamiento por lotes pueden ser un subconjunto de los datos disponibles en el servidor de procesamiento en línea.

Tipo de réplica que se debe utilizar en este escenario:
SQL Server utiliza una metáfora del sector editorial para describir los componentes del sistema de réplica. Los componentes incluyen el publicador, los suscriptores, las publicaciones y artículos, y las suscripciones.

- En el primer diagrama anterior, el servidor de procesamiento en línea es el publicador. Algunos o todos los datos del servidor de procesamiento en línea se incluyen en la publicación, donde cada tabla de datos es un artículo (los artículos también pueden ser otros objetos de la base de datos, como procedimientos almacenados). El servidor de procesamiento por lotes es un suscriptor de la publicación que recibe esquemas y datos como suscripción.
- Si los resultados se propagan de nuevo al servidor de procesamiento en línea, el servidor de procesamiento por lotes es también un publicador (normalmente con una publicación idéntica a la del servidor de procesamiento en línea), y el servidor de procesamiento en línea se suscribe a dicha publicación.

SQL Server ofrece diferentes tipos de réplica para distintos requisitos de aplicación: réplica de instantáneas, réplica transaccional y réplica de mezcla.
La mejor implementación para este escenario es la réplica transaccional, que se adapta perfectamente para controlar los requisitos indicados en la sección anterior.
Por diseño, la réplica transaccional satisface los requisitos principales de este escenario:

- Coherencia transaccional
- Latencia baja
- Rendimiento alto
- Sobrecarga mínima

Las opciones que se deben considerar en este caso son el filtrado, la réplica transaccional de punto a punto y la réplica transaccional bidireccional:

- La réplica transaccional permite filtrar columnas y filas, de manera que el servidor de procesamiento por lotes reciba sólo los datos requeridos por la aplicación.
- La réplica transaccional permite propagar cambios en más de una dirección utilizando la réplica de punto a punto o la opción bidireccional.

Pasos para implementar esta situación:
Para implementar esta situación, es preciso crear en primer lugar una publicación y suscripciones, y a continuación inicializar cada suscripción.

6. Resumen

Luego de un exhaustivo análisis, y un minucioso estudio acerca de la situación actual del Procesamiento por Lote, se han expuesto de manera organizada los conocimientos más relevantes encontrados durante el proceso investigativo del desarrollo de este trabajo, con el objetivo de brindar toda la información necesaria en términos de conceptos, definiciones y conocimientos previos ya existentes.

Capítulo 2. Propuesta de Solución

1. Spring Batch.

Uno de los componentes de Spring más conocidos hoy en día. Esta no será una guía completa del Framework, aunque para facilitar el primer contacto con este componente se nombrarán algunas de sus características.

2. Características

Para facilitar el diseño e implementación de sistemas por lotes, los bloques estructurales (módulos de programación) y diseños, le deben ser facilitados a los diseñadores y programadores en forma de gráficos de modelo estructural (diagramas de flujo) y Shell (código encapsulado, estructuras, clases, etc.). Cuando se comienza a diseñar un trabajo por lotes, la lógica del negocio debe ser descompuesta en una serie de funcionalidades que puedan ser implementados usando los siguientes estándares de bloques estructurales:

- **Aplicaciones de Conversión:** Por cada tipo de archivo suministrado o generado por un sistema externo, es necesaria la creación de una aplicación de conversión para traducir adecuadamente las transacciones de registros suministrados a un formato estándar requerido para el procesamiento. Estas aplicaciones de conversión pueden ser total o parcialmente módulos de utilidades de traducción. (ver Servicios de Lote Básico, “Basic Batch Servicies”).
- **Aplicaciones de Validación:** Las aplicaciones de validación garantizan que todos los registros de entrada/salida sean correctos y consistentes. La validación se ubica normalmente en los encabezados y pies (Headers y Trailers) de los archivos, resúmenes de errores y algoritmos de validación así como pruebas de referencia cruzada a nivel de registro.
- **Aplicaciones de Extracción:** Una aplicación que lee un grupo de registros desde una base de satos o un archivo de entrada, selecciona registros basándose en reglas predefinidas, y los escribe en un archivo de salida.
- **Aplicaciones de Extracción/Actualización:** Una aplicación que lee registros desde una base de datos o un archivo de entrada, y realiza cambios en una base de datos o archivo de salida determinados por los datos encontrados el cada registro de entrada.
- **Aplicaciones de Procesamiento y Actualización:** Una aplicación que realiza procesamiento sobre registros de entrada provenientes de una aplicación de extracción o validación. El procesamiento normalmente involucra el acceso a una base de datos para obtener datos requeridos para el procesamiento, potencialmente actualizando la base de datos y creando registros para procesamiento externo.

- **Aplicaciones de Salida/Formato:** Aplicaciones que leen un archivo de entrada, reestructuran los datos obtenidos de acuerdo a estándares de formato, y producen un archivo de salida para imprimir o transmitir a otro programa o sistema.

Además deberá ser suministrado un Shell de aplicación básico (aquí Shell debe leerse como estructura, arquitectura) para la lógica del negocio que no pueda ser construida usando los estándares anteriormente mencionados. En adición a los bloques estructurales principales, cada aplicación puede emplear uno o más estándares de funcionalidad, como son:

- Ordenar (Sort): Un programa que lee un archivo de entrada y produce otro de salida donde los registros son reorganizados utilizando un campo clave dentro de ellos. El ordenamiento es normalmente realizado por sistemas de herramientas estándares.
- Separar (Split): Un programa que lee un archivo de entrada y escribe cada registro en uno de los distintos archivos de salida basado en los valores de un campo determinado. La separación puede ser controlada o realizada por una herramienta de control por parámetros estándar.
- Unir (Merge): Un programa que lee desde múltiples archivos de entrada y produce un único archivo de salida combinando los datos provenientes de las entradas. La unión puede ser controlada o realizada por una herramienta de control por parámetros estándar.

Las Aplicaciones de Lote pueden ser categorizadas por su fuente de entrada:

- Aplicaciones controladas por bases de datos, son reguladas por valores o filas obtenidos desde la base de datos.
- Aplicaciones controladas por archivos, son reguladas por valores y registros obtenidos desde un archivo.
- Aplicaciones controladas por mensajes, son reguladas por mensajes recibidos desde una cola.

El comienzo de cualquier sistema de lotes es la estrategia de procesamiento. Los factores que afectan la selección de la estrategia incluyen: volumen de lotes estimado, concurrencia con un sistema de lotes on-line o de otro tipo, ventanas de lote existentes y cualquier otra iniciativa.

3. Principios del diseño de la Aplicación y la Base de Datos

Una arquitectura que soporte aplicaciones multi-particionadas que corren contra bases de datos particionadas usando la propuesta de campo llave, deberá incluir un almacén (depósito) central de particiones para almacenar los parámetros de partición. Esto provee flexibilidad y le asegura mantenimiento. El almacén generalmente consistirá en una tabla simple conocida como tabla de partición.

La información contenida en la tabla de partición será estática y por lo general deberá ser mantenida por la DBA (Data Base Application). La tabla consistirá en una fila por cada partición de una aplicación multi-particionada. La tabla deberá tener columnas para: Código ID del Programa, Número de Partición (ID lógico de la partición), Valor Inferior del campo llave para esta partición, Valor Superior del campo llave para esta partición.

Al iniciar, el id del programa y el número de partición deberán ser pasados a la tabla desde la arquitectura (permiso de Control de Procesamiento de Tarea, Control Processing Task let). Estas variables son usadas para la lectura de la tabla de partición, para determinar que rango de datos la aplicación va a procesar (si la propuesta campo llave es usada). En adición el número de partición se usará durante el procesamiento para:

- Agregar a los archivos/bases de datos de salida, actualizaciones para que el proceso de unión funcione correctamente.
- Reportar el procesamiento normal al log del lote y cualquier error que pueda ocurrir durante la ejecución del manejador de errores (error handler) de la aplicación.

3.1. Minimizando "Cierres Fatales"

Cuando las aplicaciones corren en paralelo o particionadas, pueden ocurrir conflictos en las bases de datos y "cierres fatales". Es esencial que el equipo de diseño elimine las situaciones de conflicto tanto como sea posible, en el propio diseño de la base de datos.

Además se asegure que los índices de las tablas sean diseñados con la prevención en mente de "cierre fatales".

Puntos críticos y “cierre fatales” ocurren frecuentemente en tablas de administración o arquitectura tales como las tablas de log, control y de protección. Las implicaciones de esto deben ser tomadas en cuenta también. Una prueba de estrés realista es crítica para determinar los posibles cuellos de botella en la arquitectura.

Para minimizar el impacto de los conflictos en los datos, la arquitectura deberá proveer servicios como espera-y-reintento (wait-and-retry) cuando se accede a una base de datos o cuando aparece un “cierre fatal”. Esto implica un mecanismo incrustado (built-in) para responder a ciertos códigos de retorno de la base de datos y como alternativa de levantar un manejador de errores inmediato.

3.2. Validación y Entrada de Parámetros

La arquitectura de la partición debe ser relativamente transparente a los desarrolladores de la aplicación. La arquitectura que debe ejecutar todas las tareas asociadas con la ejecución en modo particionado incluyen:

- Obtener los parámetros de la partición antes del inicio de la aplicación.
- Validar los parámetros de la partición antes del inicio de la aplicación.
- Entrar los parámetros a la aplicación al iniciar.

La validación debe incluir chequeos para asegurar que:

- La aplicación tenga particiones suficientes para cubrir todos los datos.
- No existan huecos entre las particiones.

Si la base de datos está particionada, es necesaria alguna validación adicional para garantizar que una sola partición no abarque todas las particiones de la base de datos.

4. Arquitectura de Spring Batch

4.1. Arquitectura Básica

Spring Batch fue diseñado con extensibilidad y pensando en un diverso grupo de usuarios finales. La figura muestra un esquema de las capas de arquitectura que soportan la extensibilidad y el fácil manejo por los usuarios o desarrolladores finales.

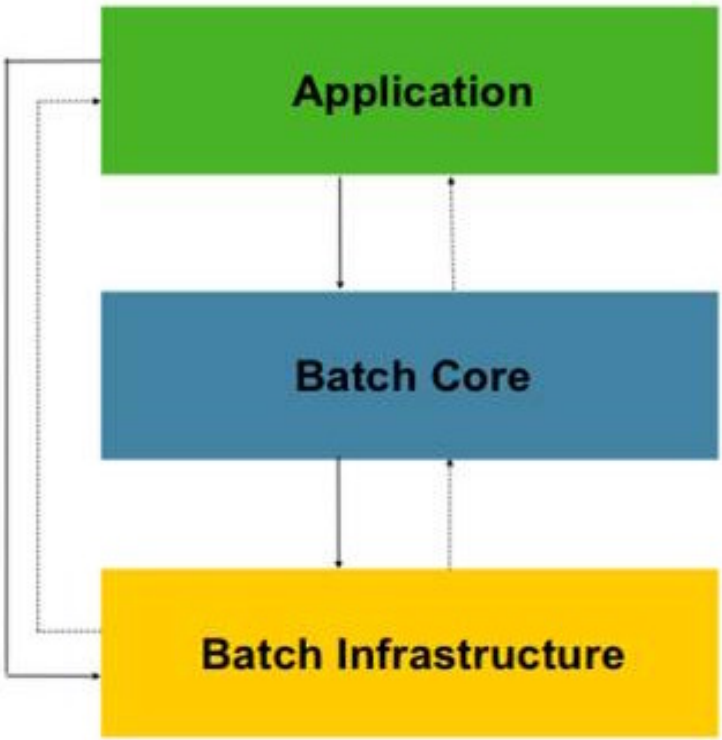

Ilustración 5: Capas de la Arquitectura de Spring Batch

Esta arquitectura destaca tres componentes de mayor nivel: Aplicación (Application), Núcleo (Core), e Infraestructura (Infraestructure).

- La aplicación contiene todos los trabajos de lote y el código personalizado por los desarrolladores usando Spring Batch.
- El Núcleo contiene las clases de tiempo de ejecución necesarias para efectuar y controlar un trabajo de lote. Incluye elementos tales como las implementaciones de: JobLauncher, Job y Step.

- Tanto Aplicación y Núcleo están construidas sobre una infraestructura común. Esta contiene lectores y escritores comunes, y servicios como RetryTemplate, que son usados tanto por los desarrolladores de aplicación como por el núcleo del Framework en sí mismo.

4.2. Procesamiento

4.2.1. Orientado a Objetos

Inicialmente la estrategia de procesamiento brindada por Spring Batch fue orientada a objetos:

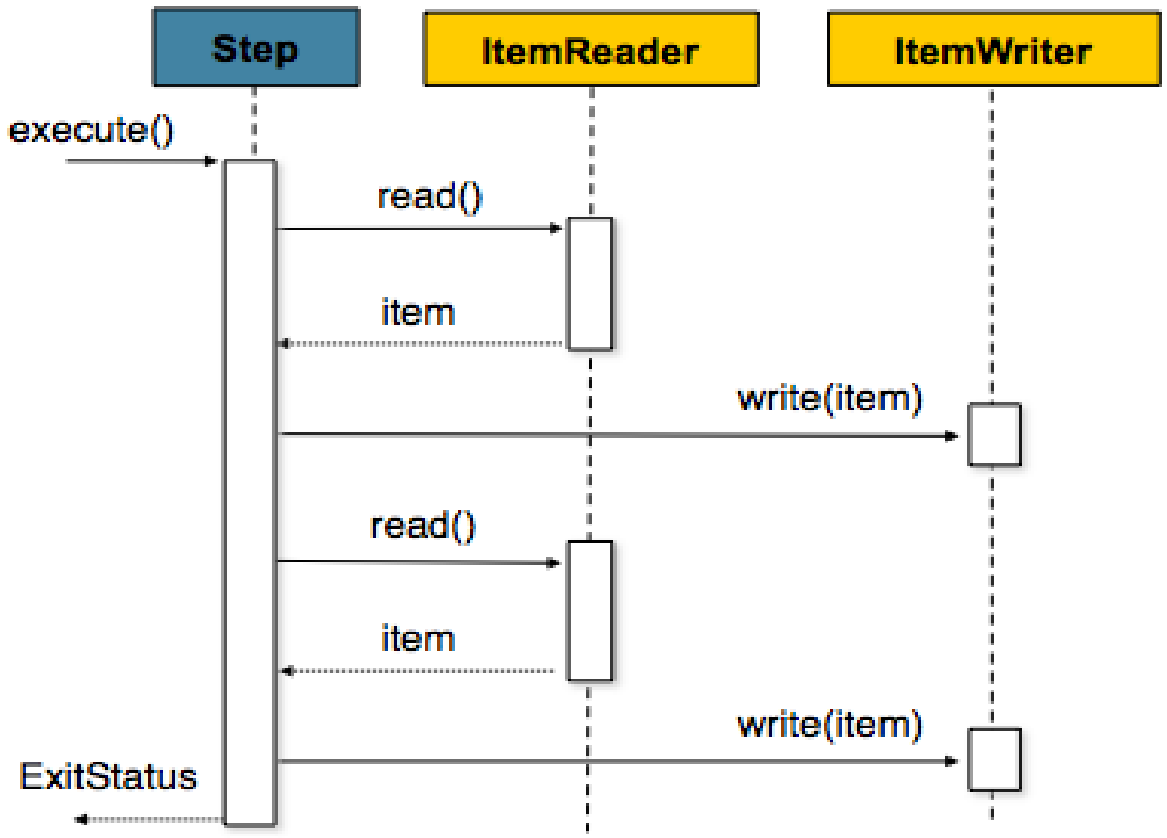

Ilustración 6: Estrategia Orientada a Objetos

En un procesamiento orientado a objetos, el ItemReader devuelve un Object (objeto) que es manejado posteriormente por el ItemWriter, depositando periódicamente cuando los objetos disparan el intervalo de deposición. Por ejemplo, si el intervalo de deposición es 5, el ItemReader y el ItemWriter serán llamados 5 veces. Esto será demostrado en el siguiente ejemplo de código:

```
for (int i = 0; i < commitInterval; i++)
{
 Object item = itemReader.read ();
 itemWriter.write (item);
}
```

Ambas interfaces, lectura y escritura, fueron completamente ensambladas siguiendo esta propuesta:

```
Public interface ItemReader
{
 Object read () throws Exception;
 void mark () throws MarkFailedException;
 void reset () throws ResetFailedException;
}

Public interface ItemWriter
{
 void write (Object item) throws Exception;
 void flush () throws FlushFailedException;
 void clear () throws ClearFailedException;
}
```

Debido a que el rango de procesamiento era de un objeto, soportar escenarios de marcha atrás (rollback) requería de métodos adicionales, que son marcar (mark), resetear (reset), flachear (flush), y limpiar (clear).

Si, después de una lectura satisfactoria de 2 objetos, un tercero provoca un error cuando es leído, la transacción deberá ser virada atrás. En este caso, el método clear será llamado en el escritor, indicando que deberá limpiar su buffer, y reset sería llamado en el lector indicando que deberá volver a la última posición donde estaba cuando mark fue llamado (tanto mark y flush son llamados en la deposición).

4.2.2. Orientado a Bloques

Actualmente la estrategia de procesamiento brindada por Spring Batch está orientada a bloques:

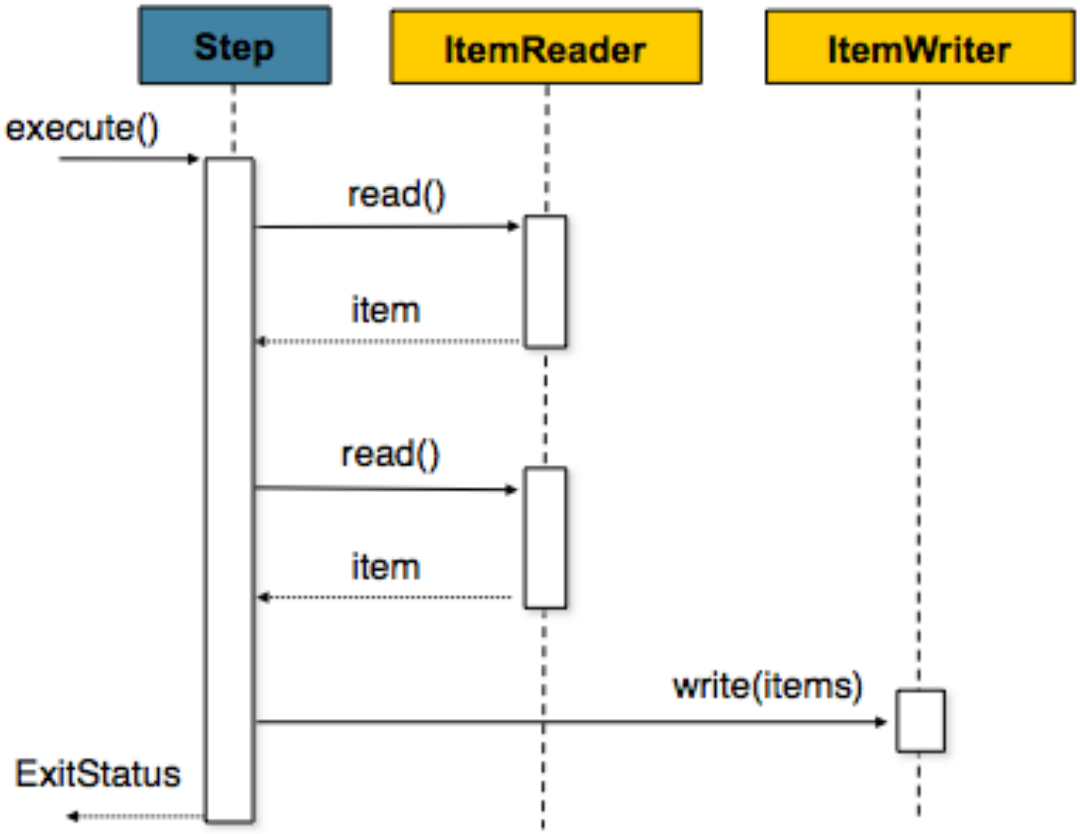

Ilustración 7: Estrategia Orientada a Bloques

Usando el mismo ejemplo anterior, y a diferencia del procesamiento orientado a objetos, si el intervalo de deposición es 5, el lector será llamado 5 veces y el escritor una. Los objetos a leer serán agregados a una lista, la que finalmente es escrita como ilustra el ejemplo siguiente:

```
List items = new Arraylist ();
for (int i = 0; i < commitInterval; i++)
{
 items.add(itemReader.read());
}
itemWriter.write (items);
```

Esta propuesta no solo permite procesamiento y escalabilidad más simple, sino también hace que las interfaces de lectura y escritura sean más limpias:

```
Public interface ItemReader<T>
{
 T read () throws Exception, UnexpectedInputException, ParseException;
}

Public interface ItemWriter<T>
{
 void write (List<? extends T> items) throws Exception;
}
```

Como se puede apreciar, las interfaces ya no contienen los métodos mark, reset, flush y clear. Esto hace que la creación de escritores y lectores sea menos complicada para los desarrolladores. En el caso de ItemReader, la interfaz ahora no presenta retrocesos. El Framework memorizará los objetos leídos para los desarrolladores en caso de rollback (aunque existen excepciones si el recurso señalado es transaccional), ItemWriter es simplificado también, ya que obtiene un bloque de objetos enteros de una vez, regularmente uno a la vez, puede decidir desechar cualquier recurso antes de devolver el control al Step.

Estos cambios han mejorado considerablemente la extensibilidad de la herramienta, y han aportado mayor comodidad a los desarrolladores; aunque no han sido las únicas mejoras realizadas.

4.3. Mejoras en el acceso a Metadatos

La interfaz del JobRepository (depósito de datos) representa las operaciones CRUD básicas en el trabajo con metadatos. Pero, puede ser útil consultar los metadatos.
Por esto, las interfaces del JobExplorer y el JobOperator fueron creadas:

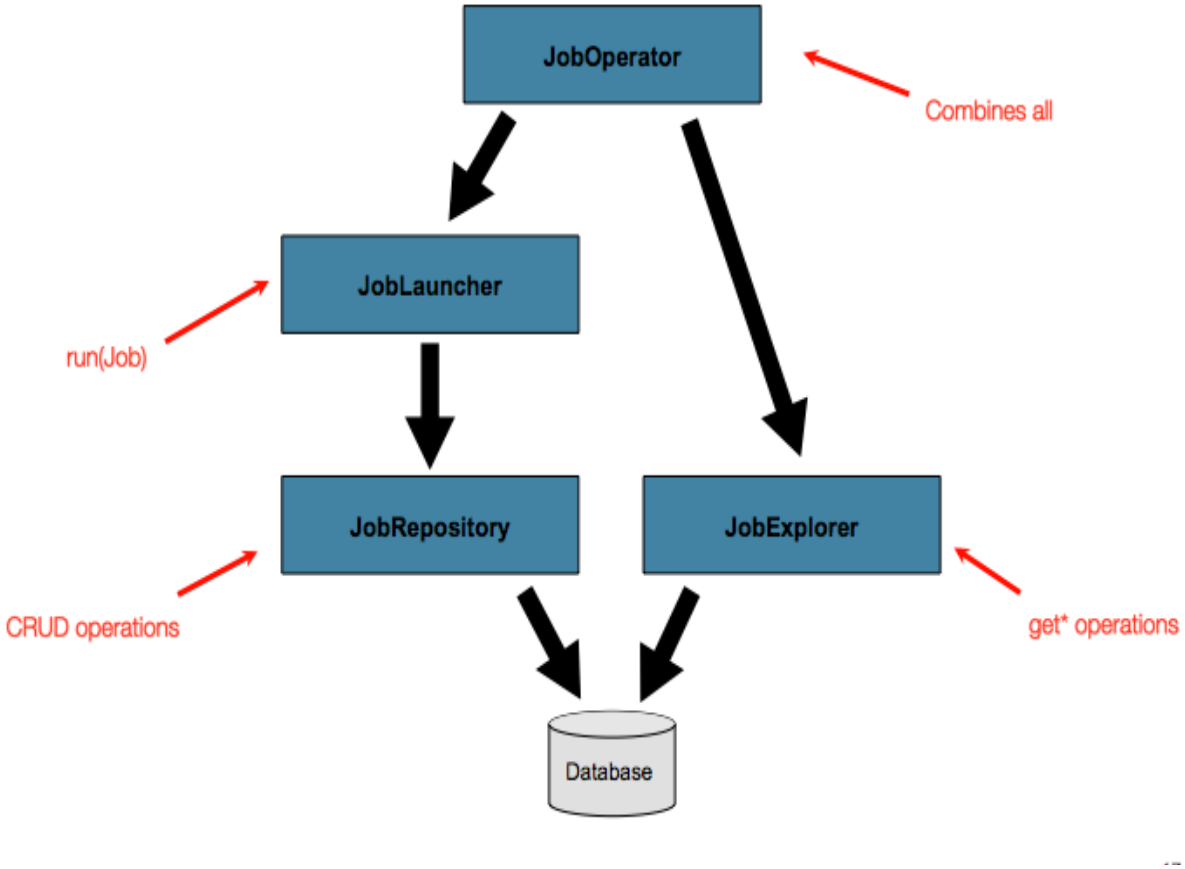

Ilustración 8: Acceso a Metadatos

4.4. Lenguaje de Dominio en el Procesamiento por Lote

Para cualquier arquitecto de lotes con experiencia, los conceptos generales del procesamiento de lotes usados en Spring Batch le serán familiares y confortables; y para los que se inician afortunadamente, el modelo de objetos lo explican los nombres por sí mismo.
Existen “Trabajos” (Jobs) y “Pasos” (Steps), además son suministradas unidades de procesamiento para desarrolladores llamadas “Lectores” (ItemReader) y “Escritores” (ItemWriter). Sin embargo, debido a los modelos, operaciones, plantillas, llamadas e idiomas de Spring existen oportunidades para lo siguiente:

- Mejoras significativas en cuanto a la separación clara de ocupaciones.
- Capas de arquitectura y servicios proveídos como interfaces claramente delineados.
- Implementaciones simples y por defecto que permiten una rápida adopción y un fácil uso fuera del marco.
- Extensibilidad significativamente mejorada.

El diagrama que se muestra a continuación es una versión simplificada de la arquitectura de referencia del lote que fue usada por décadas. Provee una idea general de los componentes que conforman el lenguaje de dominio del procesamiento por lotes.
Esta arquitectura de Framework es un diseño que ha sido probado por décadas en implementaciones de un gran número de plataformas (COBOL/Mainframe, C++/Unix, y ahora Java). Los desarrolladores que usan JCL y COBOL se sienten igual de confortables con los conceptos como los que usan C++, C# y Java.
Spring Batch brinda una implementación física de las capas, componentes y servicios técnicos comúnmente encontrados en sistemas robustos usados para lidiar con la creación de aplicaciones por lotes desde simples hasta complejas, con la infraestructura y extensiones para manejar grandes necesidades de procesamiento.

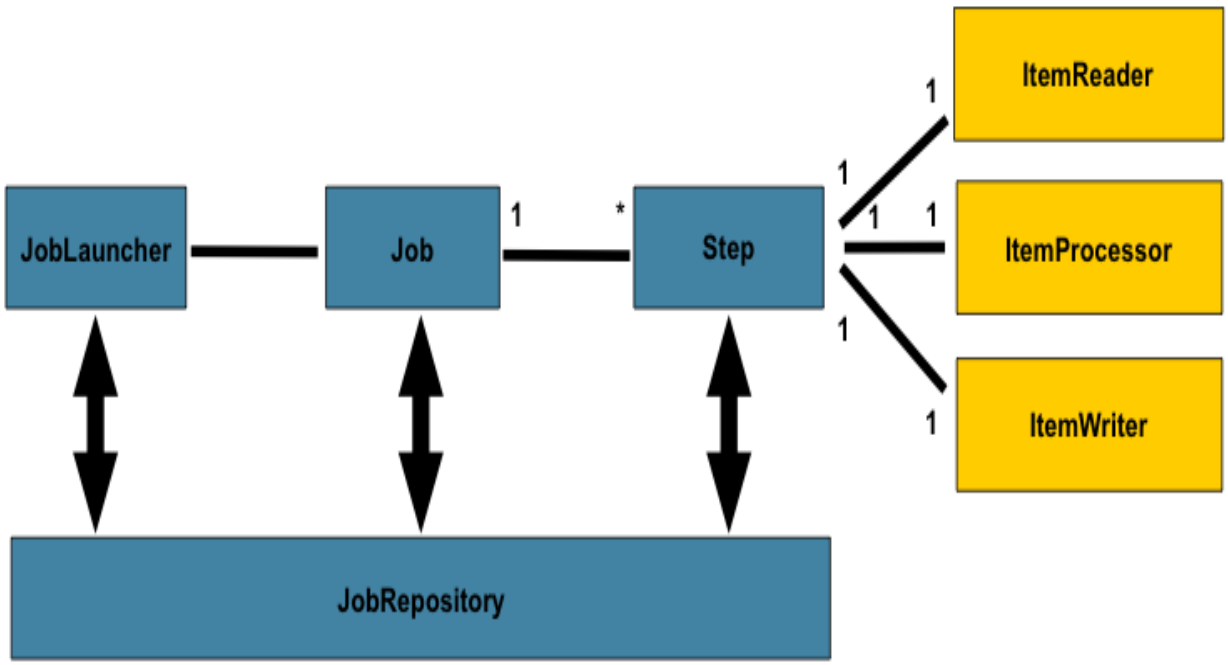

Ilustración 9: Estereotipos de Lote

Este diagrama ilustra los conceptos clave que conforman el lenguaje de dominio del lote.

Un Job (Trabajo) está compuesto de varios Step (Pasos), que se ejecutan en forma secuencial o condicional, en dependencia de cómo sea definido por los desarrolladores. Cada Step tiene exactamente un lector (ItemReader), un procesador (ItemProcessor) y un escritor (ItemWriter). El mismo Job puede configurarse con distintos parámetros que lo diferencian (por ejemplo, la fecha de ejecución), esto se conoce como (JobInstance). Un JobInstance es la particularización de un Job con un conjunto de datos determinados. La ejecución de un JobInstance es un (JobExecution). Por lo que, un JobInstance puede tener asociados muchos JobExecution, uno por cada intento de ejecución (por ejemplo, 2 intentos fallidos y uno exitoso).

Además, Spring Batch lleva un registro de todas las ejecuciones y parámetros con las que se lanzaron los Jobs. El encargado de guardar estos registros es el (JobRepository), que cuenta con una implementación para almacenar la información en una base de datos. Spring Batch utiliza un modelo de tablas propio para guardar la información de las corridas, tiempos, parámetros de invocación, pasos que se ejecutaron y otros datos. Por último, al encargado de lanzar los Jobs se le llama (JobLauncher).

4.4.1. Job

El elemento principal de Spring Batch es el Job. Un Job representa un procesamiento Batch a ejecutar, o sea es una entidad que encapsula un proceso de lote por entero.
Como es común en otros proyectos de Spring, un Job será alimentado por un archivo de configuración XML, este archivo puede ser referenciado como configuración del Job.
Sin embargo, el Job es justamente el tope de una herencia general:

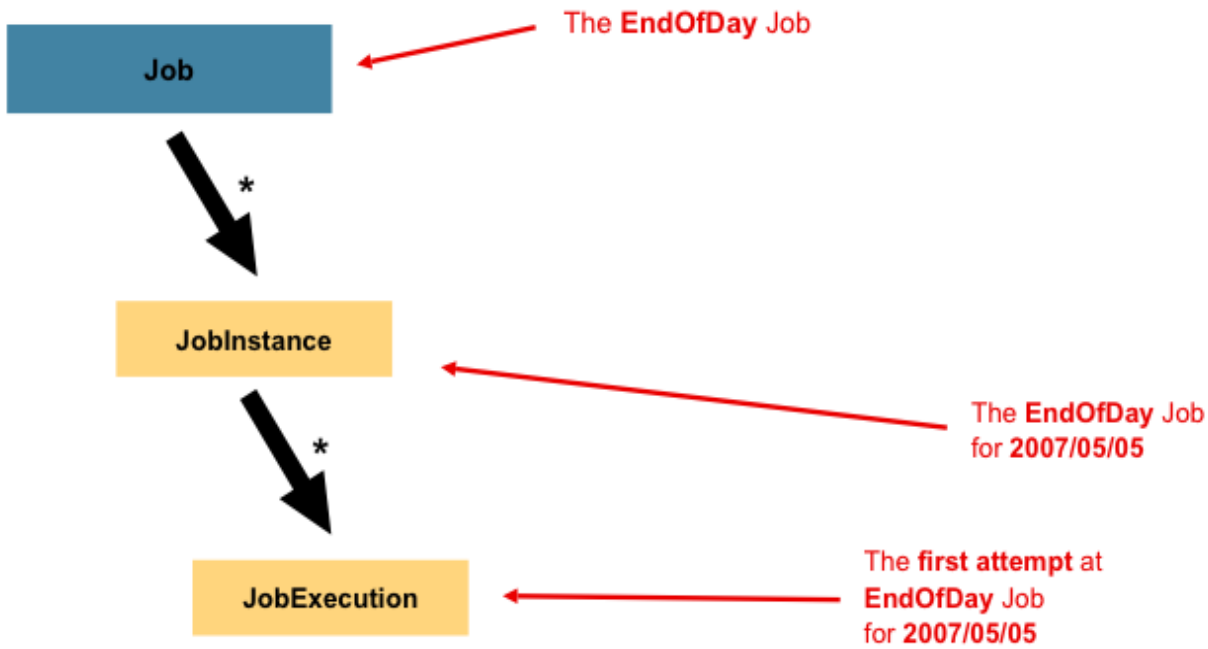

Ilustración 10: Job

En Spring Batch, un Job es simplemente un contenedor de Pasos. Combina múltiples pasos que pertenecen como unidad lógica a un flujo y permiten la configuración de propiedades globales de los pasos, como la posibilidad de ser reiniciados o no.
La configuración de un Job contiene:

- El nombre simple del Job.
- Definiciones y orden de los pasos.
- Y si el Job es reiniciable o no.

Una implementación simple es brindada por defecto en Spring Batch, en forma de la clase SimpleJob, la cual crea alguna funcionalidad estándar en el principio del Job, aunque el espacio de nombre del lote, abstrae fuera la necesidad de instanciarlo directamente. En cambio, puede ser usada la etiqueta <Job>:

```
<job id="footballJob">
 <step id="playerload" next="gameLoad"/>
 <step id="gameLoad" next="playerSummarization"/>
 <step id="playerSummarization"/>
</job>
```

4.4.2. JobInstance

Una JobInstance se remite al concepto de una ejecución lógica de un Job. Consideremos un trabajo de lote que se ejecute solo una vez al finalizar el día, tal como el Job “EndOfDay”.
Existe un Job “EndOfDay”, pero cada ejecución individual de Job deberá ser traceada por separado. En el caso de este trabajo, habrá una JobInstance por día. Por ejemplo, existirá una ejecución el día 1ro de enero y otra el día 2 de enero. Si la ejecución del día 1ro falla y es ejecutada al día siguiente, será aún la ejecución del día anterior. (Usualmente esto corresponde también con los datos que está procesando, esto es: la ejecución del Job del día 1ro de enero procesa los datos del día 1ro de enero, etc.). Entonces, cada JobInstance puede tener múltiples ejecuciones, y solo una JobInstance correspondiente a un Job en particular puede ser ejecutada en un tiempo dado. La definición de una JobInstance no tiene en lo absoluto repercusión en los datos que cargará. Es responsabilidad de la implementación del ItemReader determinar cómo los datos serán cargados. Por ejemplo, en el escenario “EndOfDay”, debe existir una columna en los datos indicando la “fecha efectiva” o el “horario efectivo” al cual pertenecen. Entonces, la ejecución del Job del día 1ro de enero solo usará datos del día 1ro y de la misma manera ocurrirá con el día 2. Debido a que esta determinación será probablemente una decisión del negocio, es dejada la decisión al ItemReader. Que se use la misma JobInstance determinará, si se usará o no el estado de la ejecución previa. Usar una nueva JobInstance significará un comienzo desde el principio, y el uso de una instancia existente significara el comienzo desde donde se abandonó el trabajo anterior.

4.4.3. JobParameters

Habiendo discutido el JobInstance y establecidas las diferencias con el Job, la pregunta natural que sigue es: "¿Cómo se distingue una JobInstance de otra?" La respuesta es: JobParameters. Un JobParameters es un juego de parámetros usados para el inicio de un Job de lotes. Pueden ser usados como identificación o incluso como referencia a datos durante la ejecución:

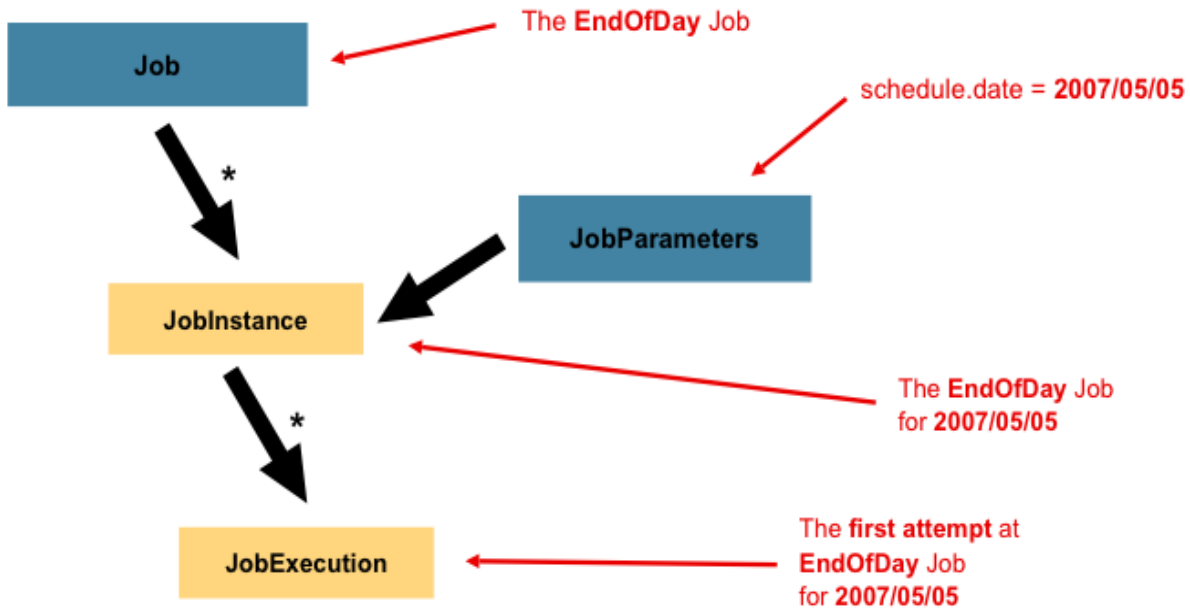

Ilustración 11: JobParameters

.

4.4.4. JobExecution

Un JobExecution se refiere al concepto técnico de un solo intento de ejecutar un Job. Una ejecución puede terminar en fallo o éxito, pero una JobInstance correspondiente a una ejecución dada no se considerará completa hasta que aquella termine exitosamente.

Retomando el ejemplo de "EndOfDay" descrito anteriormente, considere una JobInstance para 01-01-2008 que falle en el primer intento. Si vuelve a ejecutarse con los mismos parámetros de la

primera ejecución (01-01-2008), una nueva JobExecution será creada. Sin embargo, aún habrá una sola JobInstance.

Un JobExecution, es el mecanismo primario de almacenamiento para cuando ocurre un error en una ejecución, y como tal contiene muchas más propiedades que deben ser controladas de manera persistente.

Tabla 1: Propiedades del JobExecution

status	Un objeto BatchStatus que indica el estado de la ejecución. Mientras se ejecuta, es BatcStatus.STARTED, si falla, es BatchStatus.FAILED, y si finaliza correctamente, es BatchStatus.COMPLETED.
startTime	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución comenzó.
endTime	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución finalizó, sin tener en cuenta si fue satisfactoria o no.
exitStatus	El ExitStatus indica el resultado de la ejecución. Es el más importante puesto que contiene un código de salida que le será pasado al invocador.
createTime	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución fue perpetuada (se refiere persistencia) por primera vez. El trabajo puede no haber sido comenzado aún (y por tanto no tener tiempo de inicio), pero siempre habrá un createTime, que es requerido por el Framework para manejar el nivel del trabajo en ExecutionContext.
lastUpdated	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución fue perpetuada por última vez.
executionContext	La “propiedad bolsa” conteniendo cualquier dato de usuario que se necesite perpetuar entre ejecuciones.
failureExceptions	La lista de excepciones encontradas durante la ejecución de un trabajo. Estas pueden ser útiles si más de una excepción es encontrada durante el fallo de un trabajo.

4.4.5. JobRepository

El JobRepository es el mecanismo de persistencia para todos los estereotipos mencionados anteriormente. Provee operaciones CRUD para las implementaciones de JobLauncher, Job y Step. Cuando un Job es lanzado por primera vez, una JobExecution es contenida en el depósito, y durante el curso de la ejecución las implementaciones de StepExecution y JobExecution son perpetuadas pasándolas al depósito.
Un JobRepository es el encargado de almacenar información sobre las ejecuciones de los Jobs.
Existen dos implementaciones básicas de JobRepository: una que funciona en memoria (útil para desarrollo) y una que almacena la información en una base de datos.

```
<job-repository id="jobRepository"/>
```

- MapJobRepositoryFactoryBean

Esta implementación almacena la información en memoria, en un Map. Es la forma más simple de configurar un JobRepository, ya que no necesita de una base de datos.
Su configuración:

```
<beans>
  <bean id="transactionManager"

class="org.springframework.batch.support.transaction.ResourcelessTransact
ionManager"/>

  <bean id="jobRepository"

class="org.springframework.batch.core.repository.support.MapJobRepository
FactoryBean">

   <property name="transactionManager" ref="transactionManager"/>

  </bean>
</beans>
```

- JobRepositoryFactoryBean

Esta es la implementación más usada para un JobRepository, y utiliza un modelo de datos propio de Spring Batch para almacenar la información de las ejecuciones en una base de datos.

Los scripts del modelo de datos para distintas bases de datos se pueden encontrar en la raíz del archivo spring-batch-core-XXXXX.jar. Su configuración requiere, por lo tanto, de un DataSource asociado.

En el siguiente ejemplo se utiliza un DataSource de una base de datos Apache Derby:
Su configuración:

```
<beans>
  <bean id="dataSource"

class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">

    <property name="driverClassName"

value="org.apache.derby.jdbc.ClientDriver"/>

    <property name="url"

value="jdbc:derby://localhost:1527/springbatch"/>

    <property name="username" value="springbatch"/>

    <property name="password" value="springbatch"/>

  </bean>

  <bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

    <property name="dataSource" ref="dataSource" />

  </bean>

  <bean id="jobRepository"

class="org.springframework.batch.core.repository.support.JobRepositoryFac
toryBean"

    p:databaseType="derby"
    p:dataSource-ref="dataSource"
    p:transactionManager-ref="transactionManager"/>

</beans>
```

4.4.6. JobLauncher

Un JobLauncher representa una interfaz simple para el lanzamiento del Job con un juego dado de JobParameters:

```
public interface JobLauncher
{
 public JobExecution run (Job job, JobParameters jobParameters)
 throws JobExecutionAlreadyRunningException, JobRestartException;
}
```

4.4.7. Step

Un Step es el objeto del dominio que encapsula una fase independiente, secuencial o condicional, de un trabajo de lote, todo Job está compuesto por uno a más pasos (Steps). El Step contiene toda la información necesaria para controlar y definir el procesamiento actual de un lote. Esta es una descripción vaga debido a que el contenido de un Step queda a discreción del desarrollador, ya que puede ser tan simple o complejo como el desarrollador desee.

Un Step simple puede cargar datos desde un archivo hacia una base de datos requiriendo poco o ningún código (dependiendo de la implementación usada).

Un Step más complejo puede tener complicadas reglas del negocio que son aplicadas como parte del procesamiento.

Como ocurre con un Job, un Step tiene una StepExecution individual que corresponde con una única JobExecution:

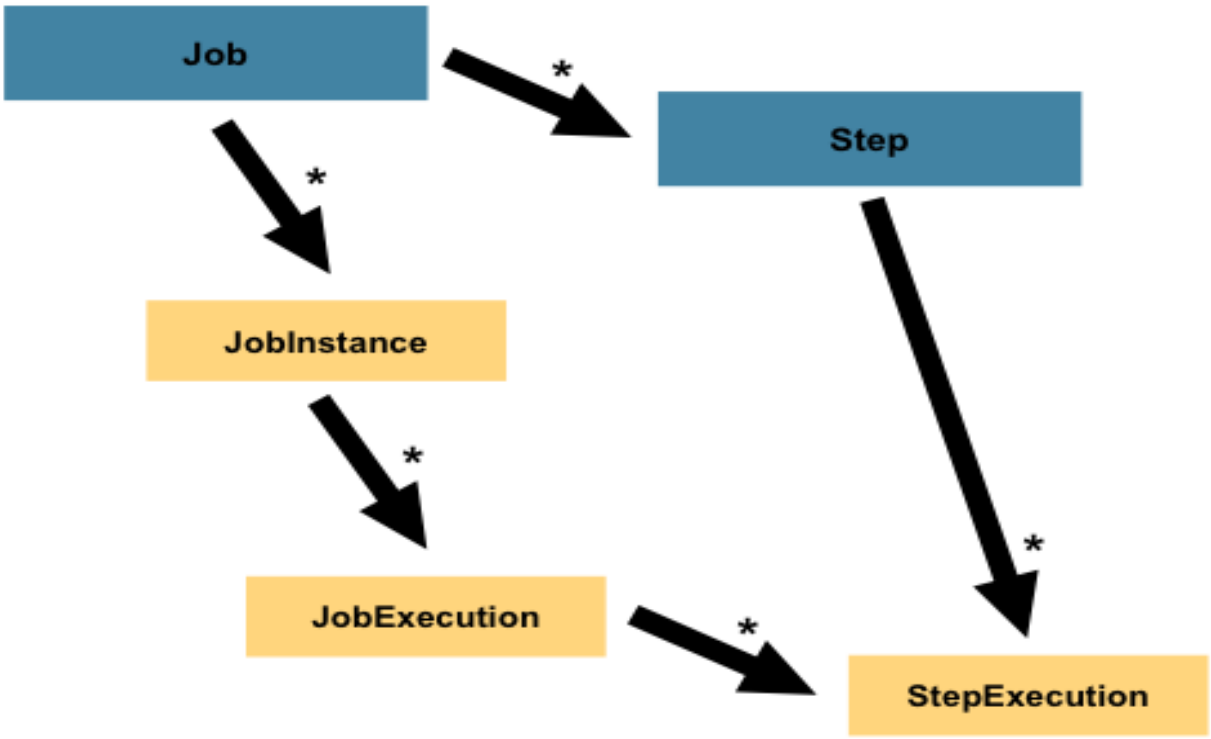

Ilustración 12: Step

4.4.8. StepExecution

Una StepExecution representa un solo intento de ejecución de un Step. Una nueva StepExecution será creada cada vez que un Step sea ejecutado, similar al JobExecution. Sin embargo, si un paso falla en su ejecución porque su paso anterior falla, no habrá ejecución persistente para él. Una StepExecution solo será creada cuando su Step sea realmente iniciado.

Las ejecuciones de pasos son representadas por objetos de la clase StepExecution. Cada ejecución contiene una referencia a su Step y JobExecution correspondientes, y datos relacionados con la transacción como los tiempos de inicio y fin, conteo de commit (ejecutar), y rollback (deshacer, vuelta atrás).
En adición, cada ejecución de pasos tendrá un ExecutionContext, que contendrá cualquier dato que el desarrollador necesite perpetuar durante la ejecución de lote, como estadística o información de estado necesaria para el reinicio.

Tabla 2: Propiedades del StepExecution

status	Un objeto BatchStatus que indica el estado de la ejecución. Mientras se ejecuta, es BatcStatus.STARTED, si falla, es BatchStatus.FAILED, y si finaliza correctamente, es BatchStatus.COMPLETED.
startTime	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución comenzó.
endTime	Un java.util.Date representando el tiempo actual del sistema cuando la ejecución finalizó, sin tener en cuenta si fue satisfactoria o no.
exitStatus	El ExitStatus indica el resultado de la ejecución. Es el más importante puesto que contiene un código de salida que le será pasado al invocador.
executionContext	La “propiedad bolsa” conteniendo cualquier dato de usuario que se necesite perpetuar entre ejecuciones.
readCount	El número de elementos que han sido leídos satisfactoriamente.
writeCount	El número de elementos que han sido escritos satisfactoriamente.
commitCount	El número de transacciones que han sido ejecutadas para esta ejecución.
rollbackCount	El número de veces que se le ha dado rollback a las transacciones del negocio controladas por el Step.
readSkipCount	El número de veces que read ha fallado, resultando en un elemento saltado (skipp).
processSkipCount	El número de veces que process ha fallado, resultando en un elemento saltado.
filterCount	El número de elementos que han sido filtrados por el ItemProcessor.
writeSkipCount	El número de veces que write ha fallado, resultando en un elemento saltado.

4.4.9. ItemReader

ItemReader es una abstracción que representa la obtención de datos para un Step, un elemento a la vez. Cuando ItemReader ha agotado los elementos que puede proveer, indicará esto devolviendo null.

4.4.10. ItemWriter

ItemWriter es una abstracción que representa las salidas de datos para un Step, un elemento a la vez. Generalmente, un escritor no tiene conocimiento de la próxima entrada que recibirá, solo el elemento que le ha sido pasado en la ejecución actual.

4.4.11. ItemProcessor

ItemProcessor es una abstracción que representa el mecanismo de procesado de un elemento. Mientras el ItemReader lee un elemento, y el ItemWriter lo escribe, el ItemProcessor provee acceso para transformar o aplicar otras reglas de procesamiento. Si, mientras es procesado el elemento, se determina que no es válido, la devolución de null indica que el elemento no debe ser escrito.

Existen muchos casos en que los objetos deben ser transformados antes de ser escritos, esto puede ser logrado usando un modelo compuesto:

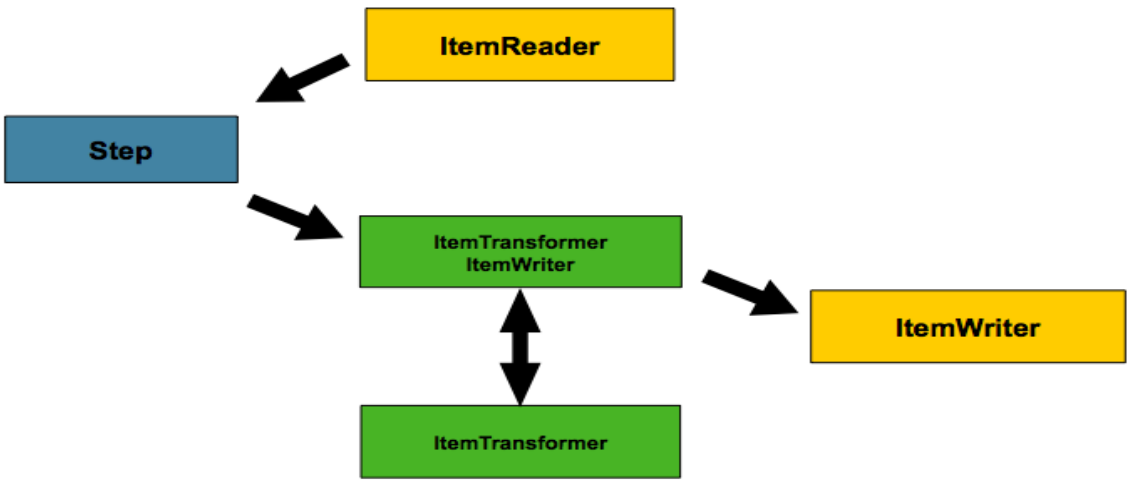

Ilustración 13: ItemTransformer

Esta propuesta funciona. Pero, requiere de una capa extra entre el lector o el escritor y el Step. Además, el ItemWriter necesitará ser registrado separadamente en un ItemStream con el Step. Por esta razón, el ItemTransformer fue renombrado a ItemProcessor y elevado al mismo nivel de ItemReader e ItemWriter.

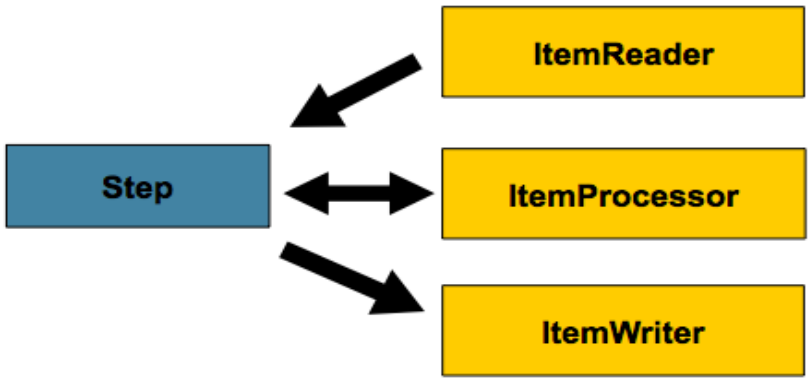

Ilustración 14: ItemProcessor

5. Un Hola Mundo con Spring Batch

Vamos entonces a realizar el clásico "Hola, Mundo" con Spring Batch. Básicamente crearemos un Job que contendrá 3 Steps:

- imprimir "Hola, "
- imprimir "Mundo"
- imprimir "!!!"

En código deberemos programar dos archivos:

- *spring-batch-demo.xml*, el contexto de Spring donde configuraremos Spring Batch, los Jobs y demás beans.
- *ImprimirTasklet.java*, que será la clase con la lógica para imprimir por consola un mensaje cualquiera.

5.1. Configuración básica

Por cada Job, vamos a utilizar un bean de Spring separado que lo representa. Hay también una serie de objetos comunes que vamos a necesitar usualmente. Vamos a ir a través de estos objetos comunes:

1- **JobLauncher:**

- Los JobLauncher son responsables de iniciar un trabajo con determinados parámetros. Existe una implementación prevista, *SimpleJobLauncher*, que se basa en una TaskExecutor para poner en marcha los trabajos. Si no específico TaskExecutor, se setea entonces un SyncTaskExecutor para utilizarlo.

2- **JobRepository:**

- Un JobRepository es el encargado de almacenar información sobre la corrida de los Jobs.
- Vamos a utilizar la implementación *MapJobRepositoryFactoryBean* que guarda la información de las ejecuciones en memoria.

- En una implementación real, donde se quiere guardar en forma persistente esta información, se puede usar la implementación *JobRepositoryFactoryBean* la cual utiliza una base de datos para almacenar toda la corrida. Spring Batch utiliza un modelo de datos con tablas propias para este fin.

3- **TransactionManager:**

- No es un bean propio de Spring Batch, pero lo utiliza el JobRepository para manejar las transacciones. En este ejemplo, como no accederemos a ningún medio transaccional, usaremos una implementación "dummy" del transactionManager ya provista por Spring Batch, llamada *ResourcelessTransactionManager*.

5.2. Spring-batch-demo.xml

```
 <beans>
  <bean id="transactionManager"
        class="org.springframework.batch.
               support.transaction.ResourcelessTransactionManager"/>
  <bean id="jobRepository"
        class="org.springframework.batch.
               core.repository.support.MapJobRepositoryFactoryBean">
        <property name="transactionManager" ref="transactionManager"/>
  </bean>
  <bean id="jobLauncher"
        class="org.springframework.batch.
               core.launch.support.SimpleJobLauncher">
      <property name="jobRepository" ref="jobRepository"/>
  </bean>
</beans>
```

5.3. Los Tasklets del Hola Mundo

Un *Tasklet* es un objeto que contiene cualquier lógica que será ejecutada como parte de un trabajo. Los Tasklets se construyen mediante la implementación de la interfaz Tasklet, y constituyen la forma más simple en Spring Batch para ejecutar código.

Vamos a aplicar una Tasklet que simplemente imprime un mensaje por consola:

```
public class ImprimirTasklet implements Tasklet
{
    private String mensaje;

       public String getMensaje()
       {
        return mensaje;
       }

       public void setMensaje(String mensaje)
       {
        this.mensaje = mensaje;
       }

       public ExitStatus execute() throws Exception
       {
        System.out.print(mensaje);
        return ExitStatus.FINISHED;
       }
}
```

Tengan en cuenta que al ejecutar el método devuelve un ExitStatus para indicar el estado de la ejecución de la Tasklet.

Vamos a definir nuestro primer Job ahora en el XML de la aplicación. Usaremos la implementación SimpleJob que ejecuta todos los pasos de secuencialmente. Con el fin de conectar un Tasklet a un Job, necesitamos un TaskletStep.

Es decir, a continuación agregaremos a nuestra configuración anterior:

- un SimpleJob.
- tres TaskletStep, que referencian a nuestros Tasklet.
- tres Tasklet, configurados para imprimir distintos mensajes.

```
 <bean id="trabajoBatch"
class="org.springframework.batch.core.job.SimpleJob">
    <property name="steps">
        <list>
            <bean id="primerPaso"

class="org.springframework.batch.core.step.tasklet.TaskletStep">
                <property name="jobRepository" ref="jobRepository"/>
                <property name="tasklet" ref="imprimirHola" />
            </bean>
```

```
            <bean id="segundoPaso"

class="org.springframework.batch.core.step.tasklet.TaskletStep">
                <property name="jobRepository" ref="jobRepository"/>
                <property name="tasklet" ref="imprimirMundo"/>
            </bean>
            <bean id="tercerPaso"

class="org.springframework.batch.core.step.tasklet.TaskletStep">
                <property name="jobRepository" ref="jobRepository"/>
                <property name="tasklet" ref="imprimirExclamacion"/>
            </bean>
        </list>
    </property>
    <property name="restartable" value="true" />
    <property name="jobRepository" ref="jobRepository" />
</bean>
<bean id="imprimirHola"
class="com.dosideas.springbatch.demo0.ImprimirTasklet">
    <property name="mensaje" value="Hola, " />
</bean>
<bean id="imprimirMundo"
class="com.dosideas.springbatch.demo0.ImprimirTasklet">
    <property name="mensaje" value="Mundo" />
</bean>
<bean id="imprimirExclamacion"

class="com.dosideas.springbatch.demo0.ImprimirTasklet">
    <property name="mensaje" value="!!!" />
</bean>
```

5.4. Ejecutando el Job

Ahora tenemos algo para poner en marcha la ejecución de nuestros Trabajos. Para esto crearemos un test JUnit que obtenga una instancia del JobLauncher, e inicie una corrida del Job.

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations={"classpath:/com/dosideas/springbatch/demo0/spring-batch-demo.xml"})

public class ImprimirHolaMundoTest
{
    @Autowired
    private SimpleJobLauncher launcher;
```

```
    @Autowired
    private SimpleJob job;

    @Test
    public void iniciarJob() throws Exception
       {
        JobParametersBuilder builder = new JobParametersBuilder();
        builder.addDate("Ejecucion", new Date());
        builder.addString("jobName", "Imprimir hola mundo por consola");
        JobParameters parameters = builder.toJobParameters();
        launcher.run(job, parameters);
       }
}
```

Este test ejecutará la tarea, y veremos por consola el mensaje "Hola, Mundo!!!".

Spring Batch también ofrece una clase conveniente para ejecutarse desde la línea de comandos: *CommandLineJobRunner*. En su forma más simple esta clase tiene de 2 argumentos: el XML de contexto de la aplicación que contiene el Job para poner en marcha y el id de ese Job. Naturalmente, requiere un JobLauncher que es configurado en el mismo XML.

A continuación se muestra cómo iniciar el trabajo desde la línea de comandos (necesita especificar el classpath):

```
java org.springframework.batch.core.launch.support.CommandLineJobRunner
     spring-batch-demo.xml trabajoBatch
```

Capítulo 3. Propuesta de Arquitectura

1. Introducción.

Actualmente al sistema de banca que se aplica en nuestro país se le han detectado algunas deficiencias propias de la arquitectura aplicada en su desarrollo, las cuales luego de un profundo análisis realizado por un grupo de especialistas de diferentes áreas de la economía y la informática, han pasado a constituir los pilares fundamentales sobre los que se deberá erguir el nuevo Sistema Bancario Cubano que actualmente se encuentra en desarrollo en la Universidad de Ciencias Informáticas (UCI).

A continuación este capítulo estará centrado en el desarrollo de una nueva arquitectura, con el objetivo de proveer a este nuevo Sistema Bancario los mecanismos necesarios para procesar los grandes volúmenes de información con los que deberá trabajar, corrigiéndose así algunas de las fallas que padece el Sistema actual.

2. El nacimiento de las arquitecturas.

Las operatorias bancarias nacieron como aplicaciones separadas que resolvían una operatoria en particular. El objetivo estaba puesto en poder procesar masivamente las cuentas y obtener un reflejo y control contable de las mismas. Sin duda, este objetivo fue alcanzado y permitió una fuerte expansión de la banca minorista. El resultado fue lo que hoy es conocido como estructura en “silos”, donde cada silo resuelve una operatoria y sus cuentas asociadas a ella.

Ilustración 15: Arquitectura en Silos

“La necesidad de relacionar las operatorias”

La propia expansión de la banca trajo como consecuencia la necesidad de relacionar las cuentas con los clientes, tal es el caso del débito automático, el servicio de comercio exterior, la aparición de la tarjeta de crédito, etc.

2.1. La aparición del factor de acoplamiento

Es a partir de esta necesidad de relacionamiento que hace su aparición el factor de acoplamiento, uno de los más significativos elementos que han conspirado contra la eficiencia y eficacia de los sistemas bancarios. El factor de acoplamiento se produce cuando es necesario relacionar aplicaciones aisladas. Para sincronizarlas y poder intercambiar información entre ellas, deben desarrollarse dos interfaces que convierten y compatibilizan datos en ambas aplicaciones, más el natural control del flujo de comunicación entre esas aplicaciones. Este factor tiene un comportamiento exponencial. Analicemos brevemente el comportamiento de este factor. Relacionar 2 aplicaciones requiere de tres programas distintos: dos interfaces, y un programa de control. Asumamos que el programa de control puede ser el mismo en cada unión de aplicaciones y que, una vez desarrollado, sólo se requieren dos programas para cada interrelación. Para 3 aplicaciones se han de requerir 6 interfaces; para 4 aplicaciones 12 programas; para 5 aplicaciones 20; y para N= N x (N-1).

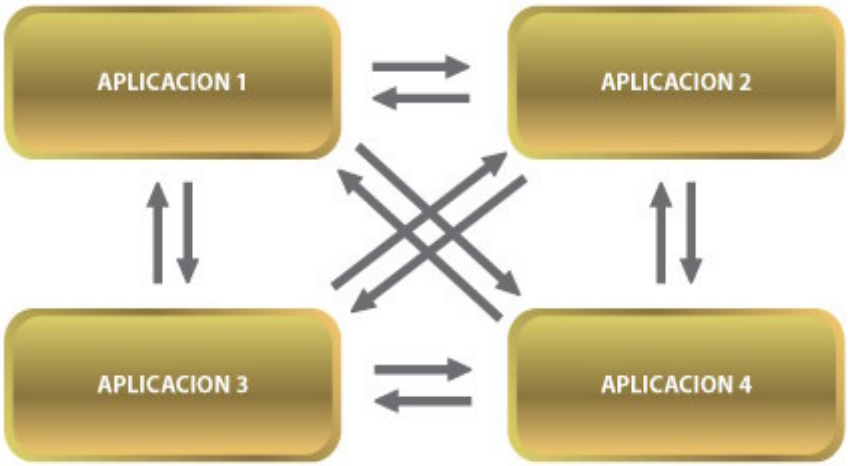

Ilustración 16: Esquema de la Formula N= N x (N-1)

Si ahora deseamos generar una nueva aplicación, debemos entonces programarla y desarrollar las interfaces que le permitan relacionarlas con las restantes. Aquí se observa una ecuación donde N = 2 x N. De esta forma, para relacionarla con cuatro aplicaciones se han de requerir 8 programas.

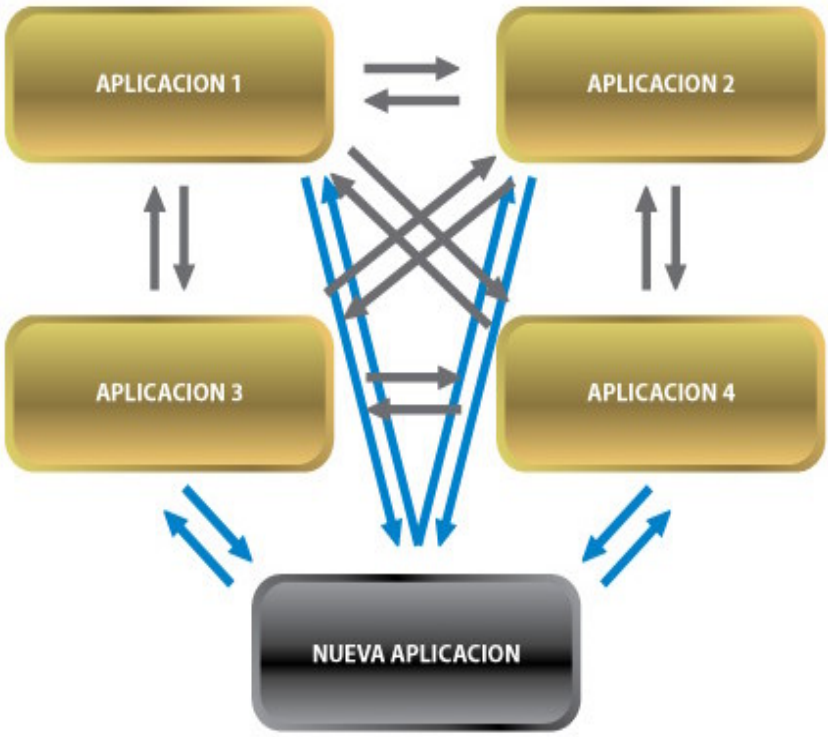

Ilustración 17: Esquema de la Formula N = 2 x N

"El acoplamiento encarece, lentifica y complica el desarrollo"

Resulta evidente que, a mayor cantidad de aplicaciones y funcionalidades que se van incorporando al sistema, más complejo, caro y lento de desarrollar se torna, sin contar con las modificaciones que el mercado obliga a realizar y que significan retocar -no sólo aquello que motiva la modificación-, sino todas las relaciones que esa parte mantiene con el resto. Aquí debe también sumarse la necesidad de testear todas las aplicaciones relacionadas para evitar la propagación de inconsistencias que suelen producirse cuando se modifica un acople. Las modificaciones se convierten en regueros de pólvora, se sabe cuando comienzan pero no dónde terminan.

2.2. El tiempo real y la reusabilidad

La aparición de lo que ha dado en llamarse “el cambio en los espacios del negocio” -en los albores del año 1995, que se tradujo en la explosión de los canales alternativos de distribución bancaria- produjo un sensible impacto sobre las arquitecturas, lo cual obligó a replantearlas.

La reacción tecnológica a los canales alternativos fue la generación de una capa de servidores con capacidad para operar 7 x 24, de manera de poder atender la demanda continua que distinguen a estos canales cuando el sistema Legacy está en procesos de consolidación, normalmente en horario nocturno. La necesidad de operar en forma independiente de los Legacy obliga a dotar a esta capa, y a cada canal, de todas las funcionalidades necesarias. Esto produce una explosión de múltiples programas que realizan idénticas funcionalidades para cada canal, además de obligar al procesamiento en paralelo para sincronizar las diferentes bases que quedan operativas cuando el Legacy sale de operación.

Las modificaciones funcionales exigen ahora no sólo la modificación y testeo de los programas centrales y sus interfaces de acoplamiento, sino también el de todos y cada uno de los canales que poseen esta funcionalidad. Igualmente ocurre con nuevos productos o procesos. La falta de la posibilidad de rehúso de las funcionalidades produce esta multiplicidad de programas diferentes que realizan la misma función. La suma de factores produce lentitud, encarecimiento y complejidad.

La suma del factor de acoplamiento y la multiplicidad de programas ha generado un incremento constante de los costos tecnológicos, sensibles demoras en la capacidad de reacción ante nuevas demandas del mercado, y una muy lenta adecuación a las necesidades de nuevos procesos de negocio.

Un impacto adicional de estas arquitecturas es la disminución de la calidad de servicio y la confiabilidad en relación al cliente, al poseer necesariamente diferentes saldos, según la hora y canal donde el cliente realice la consulta.

Este fenómeno progresivo de inadecuación de las arquitecturas tecnológicas ha convertido, en 20 años, a aquel maravilloso motor impulsor de la banca -que significó la introducción de la tecnología en el negocio- en su mayor barrera y limitación para la expansión y competitividad.

3. Arquitectura de Procesamiento por Lote para el Sistema Bancario Cubano

3.1. Procesos a ejecutar en Batch

El Sistema Bancario Cubano que se desarrolla actualmente, concentra en algunas de sus principales funcionalidades procesos muy necesarios para garantizar el correcto funcionamiento de numerosas gestiones financieras, las cuales son muy difíciles de resolver por los métodos tradicionales, debido a la gran cantidad de información que manejan. Es entonces cuando se hace imperante la necesidad de buscar nuevas vías de solución, que satisfagan las exigencias de los grandes volúmenes de información que demandan estos procesos.

Luego de un profundo análisis realizado sobre un gran número de estos procesos, se decide aplicar la técnica de Procesamiento por Lote desarrollada a lo largo de este trabajo, a los siguientes procesos:

- Proceso de Inicio del día en Banco Nacional de Cuba.
- Proceso de Cierre del día en Banco Nacional de Cuba.

3.1.1. Proceso del Inicio del Día

1. Se comprueba que el estado del sistema es de Cierre Concluido.
2. Ejecutar todos los procesos que se ejecutarán en el principio del Inicio.
3. Chequear que no haya otra persona conectada en el sistema.
4. Entrar fecha contable y cargar la fecha de posteo. Se tiene en cuenta los días no laborables (incluidos los feriados).
5. Si cambio de año, reiniciar consecutivos de referencias.
6. Reiniciar los consecutivos necesarios. Número de Transacción.
7. Salvar el Mayor del día anterior.
8. Cambiar estado del sistema a Inicio Contable.
9. Guardar Log.
10. Seleccionar Impresora.

11. Cálculo y Aplicación de Intereses.
12. Revisión de las Operaciones pendientes.
13. Cerrar cuentas pendientes a cerrarse, y las que tienen saldo 0 y no están en el Mayor.
14. Ejecutar todos los procesos que se ejecutarán al final del Inicio.
15. Emitir reportes.
16. Guardar el Diario en el Histórico.

10 – 19 Inicio contable.

20 Inicio concluido.

30 – 39 Ejecutando el cierre.

40 Cierre concluido.

50 Salva realizada.

3.1.2. Proceso del Cierre del Día

1. Verificar que se hace el inicio desde un centro contable o centro de costo.
2. Comprobar acceso.
3. Si es Centro Contable comprobar que se haya realizado el cierre en todos los centros de costo.
4. Captación de fecha del próximo inicio. Se tiene en cuenta los días no laborables (incluidos los feriados).
5. Inicio del Cierre.
6. Procesos de Validación de información. No pueden quedar transacciones fuera de línea.
7. Se guardan todas las tablas por si es necesario revertir el cierre.
8. Ejecutar todos los procesos que se ejecutaran en el principio del Cierre.
9. Devengado de intereses.
10. Ejecutar todos los procesos que se ejecutaran en el final del Cierre.
11. Emitir reportes.
12. Realizar las salvas.

3.1.3. Inicio – Cierre

Secuencia de procesos:

1. Cierre de Centro de Costo
2. Salva de Centro de Costo

} Todos los Centros de Costo

3. Cierre de Centro Contable
4. Salva de Centro Contable
5. Inicio de Centro Contable
6. Inicio de Centro de Costo

Inicio del Cierre en el Banco Nacional de Cuba:

Reglas:

1. El cierre se debe realizar por una persona con ese privilegio.
2. Solo se realiza desde una máquina.
3. Debe existir la configuración de la impresora.

- Estado del sistema = ‘030’ => Ejecución de procesos al inicio del cierre.
- Estado del sistema = ‘033’ => Ejecución de procesos al final del cierre.
- Estado del sistema = ‘034’ => Ejecución de reportes.
- Estado del sistema = ‘040’.
- Cambiar estado de los operadores a ‘030’.

Fin del Cierre en el Banco Nacional de Cuba:

Reglas:

1. No se puede realizar el cierre en el centro contable hasta que todos los centros de costo lo hayan realizado.
2. El cierre se debe realizar por una persona con ese privilegio.
3. Solo se realiza desde una maquina.
4. Se toma la próxima fecha contable teniendo en cuenta los días no laborables y los feriados.
5. Debe existir la configuración de la impresora.

6. Se salva el mayor, operaciones pendientes y el diario antes del cierre para poder compararse.

- Estado del sistema = ‘030’ => Ejecución de procesos al inicio del cierre.
- Estado del sistema = ‘031’ => Calculo de interés.
- Estado del sistema = ‘033’ => Ejecución de procesos al final del cierre.
- Estado del sistema = ‘034’ => Ejecución de reportes.
- Estado del sistema = ‘040’.
- Cambiar estado de los operadores a ‘030’.

Indicaciones para el Cierre:

- Definir como procesos de ejecución obligatoria aquellas validaciones que de ser incorrectas no pueden permitir la terminación del cierre. En este caso si el proceso detecta error en la validación debe informarlo al cierre de forma que el mismo quede detenido (variable WRET_PROCES <> 0).

  Ejemplo de validaciones a incluir en este mecanismo:

  …Cuadre Mayor-Histórico.

  …Prueba de carteras.

  …Coherencia de saldos.
- El resto de las validaciones pueden definirse como procesos no obligatorios o tablas según sea necesario.

Inicio en el Banco Nacional de Cuba:

Reglas:

1. Los centros de costo solo pueden realizar el inicio contable después de que el centro contable correspondiente lo hayan realizado.
2. El inicio se debe realizar por una persona con ese privilegio.
3. Solo se realiza desde una maquina.
4. Debe existir la configuración de la impresora.
5. Se cambia el estado de los operadores a 010.

- Estado del sistema = ‘050’:

6. Si hay cambio de año: Resetear el consecutivo de cheques y el consecutivo de registros en el M_REGIST.
7. Resetear al consecutivo de números de transacciones.

- Estado del sistema = ‘010’

8. Copiar mayor al cierre de ayer.
9. Resetear al consecutivo de números de transacciones.

- Estado del sistema = ‘010’

- Inicio del Inicio Contable

- Estado del sistema = ‘011’ => Ejecución de procesos al inicio del inicio.
- Estado del sistema = ‘013’ => Aplicación de intereses de las cuentas de ahorro.
- Estado del sistema = ‘014’ => Aplicación de intereses de las cuentas de cliente.
- Estado del sistema = ‘015’ => Revisión del Diario.
- Estado del sistema = ‘016’ => Cierre de cuentas
- Estado del sistema = ‘017’ => Ejecución de procesos al final del inicio.
- Estado del sistema = ‘018’ => Ejecución de tablas diarias.
- Estado del sistema = ‘019’ => Salvar el diario a un histórico de transacciones.
- Estado del sistema = ‘020’.

- Fin del Inicio Contable

Cierre en el Banco Nacional de Cuba:

- Chequeo que todos los Centros de Costo hayan realizado el cierre contable (campo FEC_ULTCIE = WFEC_CONT para todos los centros de costo en la tabla M_CIERRE).
- Chequeo de operador conectado al sistema.
- Si estado del sistema = ‘020’ => Captación de la próxima fecha contable (actualización del campo FEC_PROX en M_ESTSIS) y actualización de la tabla de días feriados.
- Chequeo de fecha anterior o igual a la última fecha contable
- Chequeo de fecha mayor que la ultima fecha contable en más de 7 días
- Setear frecuencia para emisión de tablas y procesos teniendo en cuenta la próxima fecha contable.
- Búsqueda de otra impresora si hay tablas que no usan la impresora por defecto (campo DEF_SALIDA = ‘OPRINTER’ en M_TABLAS).
- Creación de arreglo con los procesos a ejecutar según la periodicidad.

- Chequeo de cierre realizado en otra máquina.
- Salvando tablas antes del cierre (M_MAYOR -> M_MAYCIE, M_DIARIO -> M_DIACIE, M_HISTOR -> M_HISCIE)

• Inicio del Cierre Contable

- Estado del sistema = ‘030’ => Ejecución de procesos al inicio del cierre.
- Estado del sistema = ‘031’ => Calculo de interés.
- Estado del sistema = ‘032’ => Borrado de las tablas M_LOTES (incluyendo los FID) y M_BITACO.
- Estado del sistema = ‘033’ => Ejecución de procesos al final del cierre.
- Estado del sistema = ‘034’ => Ejecución de tablas diarias.
- Estado del sistema = ‘035’ => Reservado
- Estado del sistema = ‘036’ => Ejecución de tablas de periodicidad determinada.
- Estado del sistema = ‘040’.
- Cambiar estado de los operadores a ‘030’.

• Fin del Cierre Contable

- Borrado de la tabla T_IBATCH (lotes de la transacción general)

3.1.4. Validaciones generales

- Se prohíbe la contabilización desde cualquier transacción durante los procesos de:
  Cierre – Salva – Inicio.
  En este caso TEST_INEFOX retornará código de error 40.
- En el caso de la aplicación de correo para mensajería contable de quedar accidentalmente funcionando durante estos procesos debe validar este código de forma que no envié ratificación mala o altere los estados del M_MENSCE.
- En el caso de la aplicación de correo para autorización del CCCA de quedar accidentalmente funcionando durante estos procesos devolverá código de respuesta 80 (transacción denegada) al CCCA al no poder efectuar la contabilización.
  Tener en cuenta que no es posible devolver este código de error a los procesos que se estén ejecutando dentro del mismo cierre o inicio lo cuales deben poder contabilizar sin problema.

3.2. Modelo de Configuración del Proceso por Lote

El siguiente diagrama representa el Modelo de Configuración del Proceso por Lote, diseñado para ser aplicado en el Sistema Bancario Cubano que se está desarrollando actualmente. Está basado en la configuración estructural que brinda Spring Batch, y en su relación jerárquica.

- El primer nivel está encabezado por el JobLauncher, el cual será el encargado de realizar los lanzamientos de los Jobs (Trabajos), representados individualmente en el segundo nivel. El JobLauncher se encuentra directamente asociado al JobRepository (como se muestra en la Ilustración 18), desde el cual y a través de sus tres estructuras fundamentales: JobRepositoryFactoryBean, HibernateTransactionManager, y DataSource, se podrá acceder a los registros y salvas de cada uno de los Jobs lanzados por el JobLauncher. Esto permitirá conocer el estado detallado de cada una de las ejecuciones y hará posible la recuperación y re-lanzamiento de aquellas que su proceso no haya concluido satisfactoriamente.
- En el segundo nivel se encuentra el Job, este representa individualmente cada uno de los procesos a ejecutar en Batch. Un Job está compuesto por varios Steps (Pasos), los cuales pueden ejecutar una operación de lectura, procesado o escritura, mediante sus estructuras ItemReader, ItemProcessor, o ItemWriter, respectivamente; incluso pudiéndose realizar en el caso de ItemReader e ItemWriter, estas referidas operaciones de lectura y escritura, directamente sobre la base de datos, a través de sus dos sub-estructuras HibernateCursorItemReader e HibernateCursorItemWriter correspondientes. Esto permitirá dotar a las operatorias Batch de las acciones básicas que se aplica sobre la información.

En otras palabras:

“Se hará posible que cada proceso corrido en Batch, bajo los parámetros correctos y contando con los recursos necesarios sea capaz de ‘leer’, ‘pensar’, y ‘escribir’, de manera autónoma”

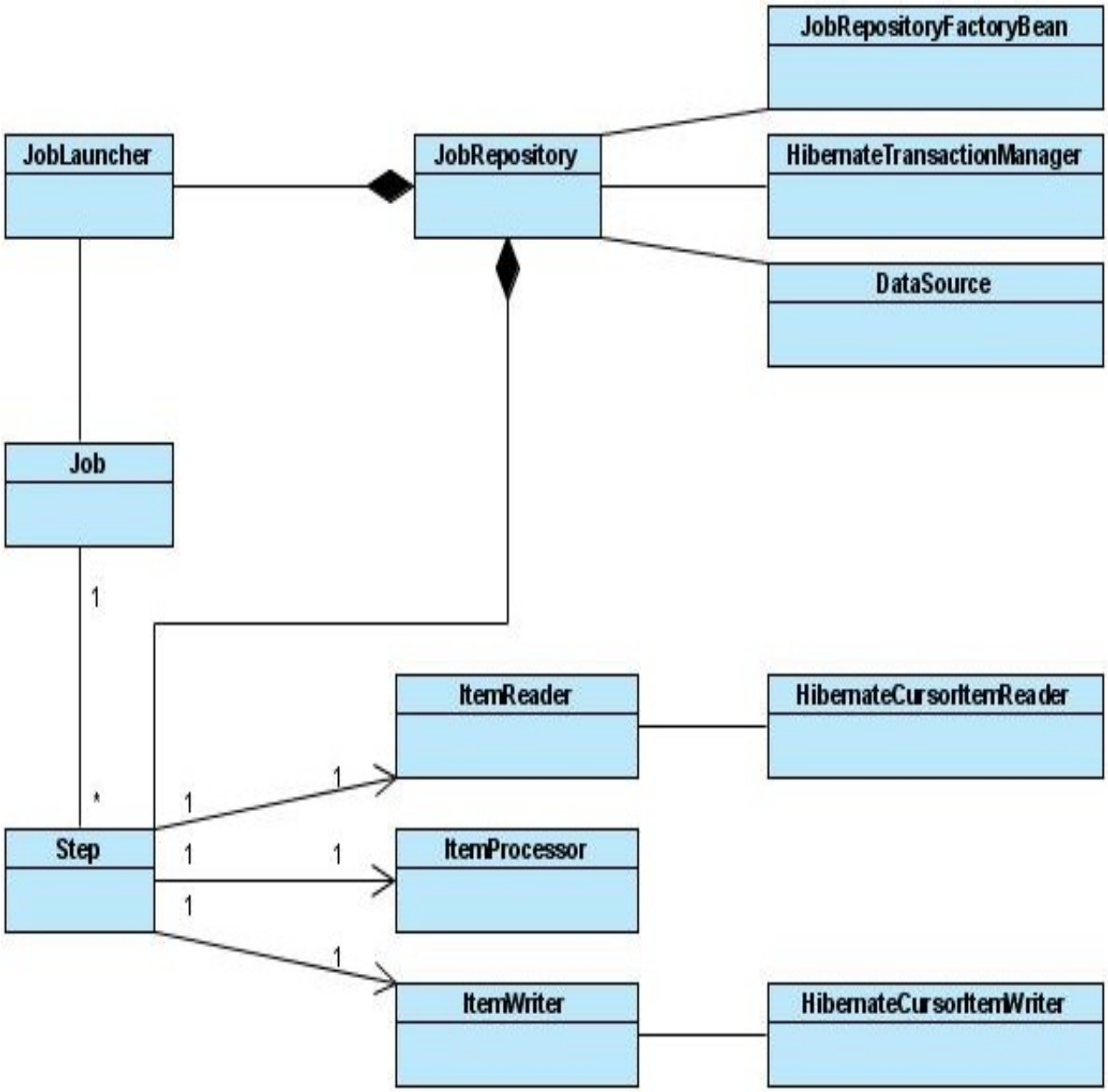

Ilustración 19: Configuración del Proceso por Lote

3.3. Modelo de Paquetes del Proceso por Lote

A continuación la figura mostrada consiste en una representación, del Modelo de Paquetes del Proceso por Lote, donde se ilustra detalladamente cómo se propone que quede distribuida dentro del Sistema Bancario Cubano, la configuración descrita con anterioridad.

- En la capa superior de paquetes, se encuentra la configuración definida con respecto al Job y su estructura interna, lo cual permitirá de una mejor manera realizar cualquier cambio en un determinado Job, y evitará consecuentemente que dicho cambio afecte las entidades persistentes en la base de datos relacionados a este Job.
- La capa inferior de este modelo de paquetes, estará concebida de manera exclusiva por el primer nivel de la configuración previamente mostrada, o sea, el nivel encabezado por el JobLauncher asociado directamente al JobRepository y su estructura interna de acceso a datos. Esto brindará la abstracción necesaria entre los procesos a ejecutar en Batch y la información almacenada en la Base de Datos, de manera que se pueda proteger eficientemente la integridad de dicha información, requisito muy recomendado en el desarrollo de aplicaciones Bancarias y Financieras. Además se podrá mantener la misma configuración general para la ejecución de los distintos Jobs, tributándole mayor flexibilidad al Sistema Bancario Cubano, ya que será posible aplicarle la misma configuración a todos los procesos que se necesiten correr en Batch, con solo definir el proceso en cuestión e insertarlo en la primera capa de paquetes.

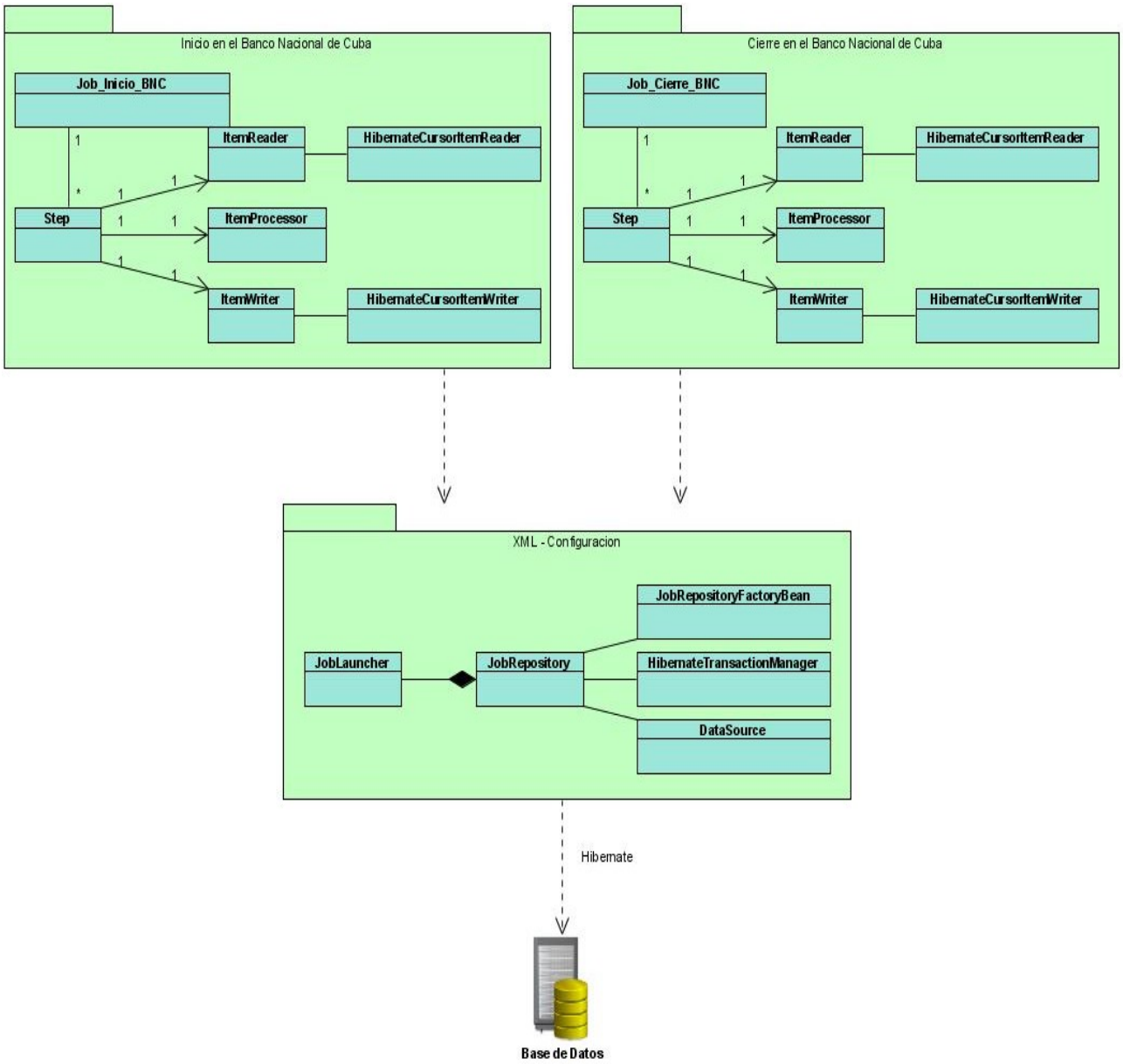

Ilustración 20: Paquetes del Proceso por Lote

Conclusiones Generales

El proceso investigativo inicialmente llevado a cabo, permitió realizar un estudio minucioso acerca de los diferentes enfoques dados en la Informática a las técnicas de Procesamiento por Lote, resultado que definió un factor decisivo en el momento de confeccionar la propuesta de solución anteriormente presentada.

Luego de reunir los conocimientos necesarios, se desarrolló una propuesta de arquitectura, la cual en su futura puesta en práctica permitirá resolver algunas de las fallas que actualmente presenta el Sistema Bancario Cubano, entre las que se encuentra el determinado problema a resolver, referente a ¿Como procesar las grandes cantidades de información que maneja el Sistema Bancario Cubano que actualmente se encuentra en desarrollo?...por lo que al ser esta arquitectura una solución puntual a dicho problema, se permite considerar satisfactoriamente cumplidos, los objetivos trazados en la fase inicial de este trabajo.

Recomendaciones

Modestamente considero necesario realizar las siguientes recomendaciones, con el objetivo de que se haga efectiva la proyección de este trabajo.

- Realizar el Análisis y Diseño correspondiente, sobre los procesos previamente definidos a procesar en lote, para facilitar su inmediata aplicación sobre el área de procesamiento de datos del Sistema Bancario Cubano, actualmente en desarrollo.
- Aplicar al presente documento las normas de calidad y autoría vigente en la Universidad de las Ciencias Informáticas (UCI), con el objetivo de hacer público su contenido y de esta manera promover el intercambio libre de conocimientos.
- Proyectar la aplicación de esta técnica de Procesamiento por Lote sobre otras áreas del Sistema Bancario Cubano, actualmente en desarrollo, para lograr mayor aprovechamiento y un mejor rendimiento de los recursos tecnológicos que se dispongan.

Finalmente se hace extensible la convocatoria a preparar al personal de desarrollo, en los temas de las disimiles técnicas de procesamiento de información existentes, para lograr una mejor calidad en el desarrollo de sistemas bancarios y financieros.

Referencias Bibliográficas

1. **R. E., Johnson.** Designing reusable classes. [book auth.] Foote B. *Journal of Object-Oriented Programming.* 1988. 1(2):22-35.
2. **González, Héctor Suarez.** Manual de Hibernate. [En línea] [Citado el: 11 de Diciembre de 2008.] http://www.javahispano.org/.
3. **Jonhson, Rod.** *Expert One-on-One Java EE Design and Development.* s.l. : Wrox Press, 2002.
4. **© 2009 Microsoft Corporation. Reservados todos los derechos.** *technet.microsoft.com.* [Online] Microsoft Corporation. [Cited: Febrero 04, 2009.] http://technet.microsoft.com/es-es/library/bb490869(en-us).aspx.
5. **© 2002 Adobe Systems Incorporated. All rights reserved.** *Adobe Photoshop.* [Online] [Cited: Febrero 06, 2009.] http://www.adobe.com/cgi-bin/aolnredirect?code=awe_571000.
6. **© 2009 Microsoft Corporation. Reservados todos los derechos.** *msdn.microsoft.com.* [Online] [Cited: Febrero 11, 2009.] http://msdn.microsoft.com/es-es/library/ms175419(SQL.90).aspx.

Bibliografía

1. http://www.nabble.com/. [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.nabble.com/How-to-use-batch-with-IBatis-td17188088.html.
2. http://www.mail-archive.com/. [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.mail-archive.com/user-java@ibatis.apache.org/msg11394.html.
3. http://www.coderanch.com/. [En línea] [Citado el: 12 de Diciembre de 2008.] http://www.coderanch.com/t/216110/Object-Relational-Mapping/java/Enabling-batch-updates-hibernate.
4. http://www.devx.com/. [En línea] [Citado el: 22 de Enero de 2009.] http://www.devx.com/tips/Tip/32115.
5. http://www.hibernate.org/. [En línea] [Citado el: 22 de Enero de 2009.] http://www.hibernate.org/hib_docs/reference/en/html/batch.html.
6. http://web.ebscohost.com/. [En línea] [Citado el: 27 de Enero de 2009.] http://web.ebscohost.com/ehost/pdf?vid=11&hid=103&sid=f98f80dd-c92f-4d54-b19b-a57723a45936%40sessionmgr3.
7. http://stackoverflow.com/. [En línea] [Citado el: 27 de Enero de 2009.] http://stackoverflow.com/questions/83093/hibernate-insert-batch-with-postgresql.
8. http://www.sqlmag.com/. [En línea] [Citado el: 27 de Enero de 2009.] http://www.sqlmag.com/Articles/Index.cfm?ArticleID=21975&DisplayTab=Article.
9. http://searchsqlserver.techtarget.com/. [En línea] [Citado el: 27 de Enero de 2009.] http://searchsqlserver.techtarget.com/tip/0,289483,sid87_gci1161826,00.html.
10. http://freeyourtech.wordpress.com/. [En línea] [Citado el: 11 de Febrero de 2009.] http://freeyourtech.wordpress.com/2008/07/15/using-postgresql-jdbc-for-bulk-updates-batch-size-vs-performance/.
11. http://www.oreillynet.com/. [En línea] [Citado el: 11 de Febrero de 2009.] http://www.oreillynet.com/pub/a/databases/2006/09/07/plpgsql-batch-updates.html?page=5.
12. http://www.devjoker.com/. [En línea] [Citado el: 11 de Febrero de 2009.] http://www.devjoker.com/html/Transferir-datos-de-SQL-Server-en-archivos-bat_843.html.
13. http://www.freedownloadmanager.org/. [En línea] [Citado el: 11 de Febrero de 2009.] http://www.freedownloadmanager.org/es/downloads/Suite_de_Sql-hornada_46311_p/.
14. http://www.javahispano.org/. [En línea] [Citado el: 17 de Marzo de 2009.] http://www.javahispano.org/search.search.action.
15. http://www.javahispano.org/. [En línea] [Citado el: 17 de Marzo de 2009.] http://www.javahispano.org/forums.thread.action?forum=2&thread=998460&id=7884360.
16. http://www.javahispano.org/. [En línea] [Citado el: 17 de Marzo de 2009.] http://www.javahispano.org/forums.thread.action?forum=2&thread=6390597&id=946320.
17. http://www.fileheaven.com/. [En línea] [Citado el: 2 de Abril de 2009.] http://www.fileheaven.com/descargar/sql-batch-suite/60720.htm.
18. http://www.hispagen.es/. [En línea] [Citado el: 7 de Abril de 2009.] http://www.hispagen.es/portal/batch.php.

19. http://www.redcientifica.com/. [En línea] [Citado el: 7 de Mayo de 2009.] http://www.redcientifica.com/doc/doc200104190004.html.
20. http://www.santafe-conicet.gov.ar/. [En línea] [Citado el: 21 de Mayo de 2009.] http://www.santafe-conicet.gov.ar/servicios/comunica/batch.htm.