UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistematización y Tratamiento de Datos Dinámicos

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

Carlos Alberto Palacios Juárez

ASESOR DE INFORME

Mtro. Juan José Carreón Granados

Ciudad Universitaria, Cd. Mx., 2018

Índice

Índice de figuras

1 Objetivo

El objetivo de este trabajo es mostrar los alcances que pueden tener el uso de tecnologías de software y su manipulación para el tratamiento de grandes cantidades de información, donde la principal tarea es la sistematización de procedimientos con el fin de lograr la reducción de errores y tiempos propios del tratamiento manual de datos.

Diseñar aplicaciones que den soporte a estructuras de datos indefinidas manejadas internamente, pues el modelo de una información puede diferir en gran medida al de otra, provocando que la información se vea afectada por sus dimensiones así como por su complejidad en significado y configuración, además existe la creciente cantidad de información que se trata por los procesos estadísticos en el Instituto, así las aplicaciones deben poder soportar estas características de la información.

Se pretende el máximo aprovechamiento de los recursos con los que se cuentan, para que el retorno de inversión se vea reflejado en el correcto uso de los bienes materiales y no materiales, que abarcan desde las redes de datos, servidores dedicados y de propósito múltiple, hasta las herramientas de software que pueden ser utilizadas a favor de una mayor capacidad de respuesta a cada bloque funcional del Instituto.

También se busca mostrar la aplicación de software para dar solución a una necesidad, donde bajo un análisis se puede determinar si un problema puede ser tratado por una de las siguientes vertientes que este trabajo considera:

- **Software Libre:** permite la obtención de código que puede ser usado y modificado sin ningún tipo de compra o licencia que limite su utilización[1], así pueden ser incrementadas sus capacidades o adaptar sus funcionalidades.
- **Software Privativo:** su uso está sujeto a una licencia que lista las condiciones bajo las cuales puede ser usado e implementado. Se adquiere bajo un costo y puede generar grandes beneficios al ajustarse a los requerimientos.
- **Desarrollo de Software:** referente a la creación de fragmentos de código para solventar una necesidad.

[1] https://www.gnu.org/philosophy/free-sw.html

2 Introducción

En este reporte se presentan las actividades realizadas en el Instituto para el área de sistemas mismas que son reflejadas a dos áreas principales, descritas en los apartados 2.1 y 2.2. Se trata de un organismo público autónomo que genera gran cantidad de información estadística y geográfica donde su sede principal está ubicada en el estado de Aguascalientes México[2], además de sedes en Ciudad de México, misma donde se crean los desarrollos mostrados en este reporte.

Al manejar grandes volúmenes de datos el Instituto tiene la necesidad de aplicar procedimientos específicos a la información que le es proporcionada por medio del levantamiento de censos, desde económicos hasta de población y vivienda, también obtiene insumos de información proveniente de otros organismos, que ofrecen datos para servir de análisis para los procesos finales.

Antes de la generación de datos estadísticos la información debe formar parte de un modelo de datos, donde cada valor y cada propiedad tienen la misma importancia que el conjunto de ellos. De esta manera los datos son tomados para aplicar análisis económicos/estadísticos que resultan en indicadores que representan el comportamiento de un agrupado de datos brutos; la información final es tomada por diferentes organizaciones públicas y privadas que hacen de su significado un parámetro en la toma de decisiones que afectan desde la organización hasta el propio país.

En el análisis de requerimientos se pudo observar las amplias soluciones que pueden ser aplicadas, es por esto que valoré las herramientas existentes para determinar si un problema puede ser resuelto mediante el uso de software privativo, de licencia libre o un desarrollo interno.

Se exponen las aportaciones realizadas en dos principales temáticas, desarrollo de software y minería de la base de datos, donde los productos finales son sistemas web basados en Java para la parte servidor[3] y Javascript para el cliente[4], además de desarrollos Java como herramientas de alimentación de base de datos.

Las aportaciones realizadas obedecen a la tarea de automatizar procesos de manera eficiente para aprovechar los recursos con los que se cuentan, esto en consideración de los grandes volúmenes de datos y la demanda de procesamiento que esta información requiere. En el Instituto existen dos elementos principales con respecto a origen y destino de los datos que son tratados en las diferentes áreas económicas e informáticas.

[2] http://www.beta.inegi.org.mx/inegi/quienes_somos.html
[3] https://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html
[4] https://www.sencha.com/products/extjs/#overview

2.1 Área generadora de información

Son las áreas del Instituto que se encargan de obtener cálculos, índices económicos y estadísticos. Por lo general trabajan con archivos excel como insumo, es decir, toman datos de archivos para realizar operaciones y ocupan el mismo recurso para exponer resultados.

2.2 Área de Sistematización

Es el área a la pertenezco y que se encarga de tomar el proceso origen de los datos para reproducir los cálculos generados por el Área Generadora de Información de manera sistemática, pues los datos en su mayoría son calculados de la misma forma, ya sea para datos pasados, presentes o futuros. Una vez sistematizado el proceso se puede reproducir para calcular los indicadores de la próxima información que un censo o una organización proporcione.

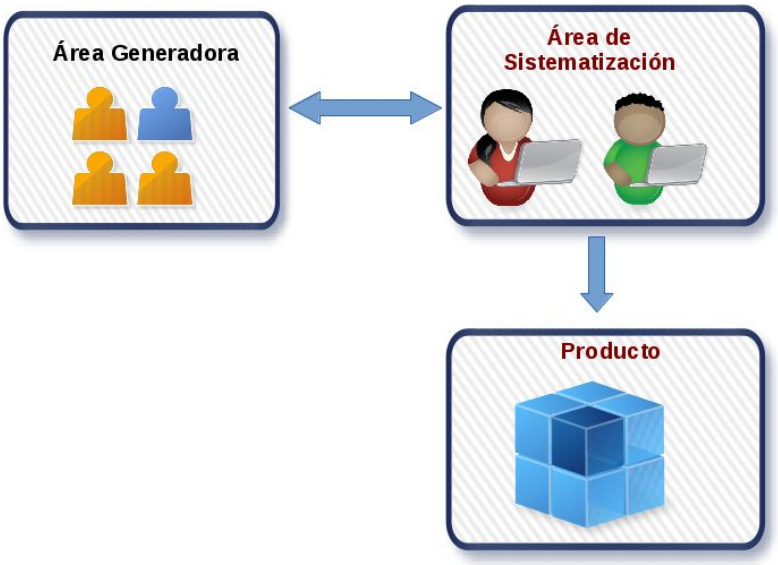

Figura 1. Relación de áreas y productos de sistematización

3 Marco teórico

3.1 Java

Es un lenguaje de programación desarrollado originalmente por Sun Microsystems, que fue iniciado por James Gosling y lanzado en 1995 como componente principal de la plataforma de Java (J2SE).

Algunas de las características del lenguaje son:

- **Orientado a Objetos:** todo es un Objeto en este lenguaje.
- **Independiente de la plataforma:** es compilado en un lenguaje independiente de la plataforma, que después es interpretado por la Máquina Virtual de Java (JVM); esto permite que se pueda ejecutar el mismo código en diferentes sistemas operativos.
- **Seguro:** permite el desarrollo de sistemas libres de virus y sin manipulación, utilizando técnicas de cifrado que lo hacen un lenguaje confiable.
- **Multihilo:** permite al lenguaje escribir programas que puedan realizar tareas simultáneas, mejorando en gran medida el rendimiento y aprovechamiento de los recursos de cómputo.

Para este trabajo se han utilizado en su mayoría los siguientes elementos de Java.

3.1.1 java.net.Socket

Es un componente dedicado a la conectividad en red. Proporciona un mecanismo de comunicación entre dos equipos, para ello se necesitan dos extremos en la comunicación, un cliente (**java.net.Socket**) y un servidor (**java.net.ServerSocket**), el primero es quien solicita una conexión y el segundo es quien acepta la solicitud de una conexión, a esta interacción se le denomina Arquitectura Cliente-Servidor.

Los siguientes pasos se producen al establecer una conexión entre dos equipos:

1. El servidor crea una instancia del objeto ServerSocket, indicando en qué puerto debe producirse, el puerto es un valor numérico que identifica el canal por el que se producirá la comunicación.
2. El servidor invoca el método accept() de la clase ServerSocket. Este método espera hasta que un cliente se conecta.
3. Después de que el servidor está esperando, un cliente crea una instancia del objeto Socket, que especifica el nombre del servidor y el número de puerto al que debe conectarse.

4. El constructor de la clase Socket intenta conectar el cliente al servidor especificado y el número de puerto, si se establece la comunicación, el cliente ahora tiene un objeto capaz de comunicarse con el servidor.
5. En el servidor, el método accept() devuelve una referencia a un nuevo socket en el servidor que está conectado al socket del cliente, de igual forma ahora tiene un objeto con el que puede comunicarse con su cliente.

Una vez realizada una conexión, la comunicación puede realizarse mediante flujos de Entrada/Salida.

3.1.2 Servicios Web (Web Services)

Son aplicaciones Cliente-Servidor que se comunican mediante Protocolo de Transferencia de Hipertexto (HTTP). Estos proporcionan un medio estándar de interoperar entre aplicaciones que se ejecutan en una variedad de plataformas.

En Java existen dos tipos de Servicios Web:

- **JAX-WS:** es la tecnología para crear servicios web que se comunican mediante XML, un formato estructurado de archivo tipo texto. El formato XML obedece a las especifiaciones definidas por SOAP, encargado de precisar la estructura, las reglas de codificación y las convenciones para representar las invocaciones y respuestas del servicio web.

- **JAX-RS:** es la tecnología para crear servicios web Representational State Transfer (RESTful). REST es adecuado para escenarios básicos que pueden trabajar con la utilización mínima de herramientas, su utilización es de poco costo en cuanto a recursos de los equipos.

3.1.3 java.lang.Thread

Son instancias de procesamiento que son ejecutadas en un tiempo determinado, la ejecución administrada de procesos e hilos respalda la definición de ejecución concurrente. También son llamados procesos ligeros, debido a que tiene un menor costo en recursos la creación de un hilo comparando con la creación de un nuevo proceso, un proceso es una instancia más completa de procesamiento, pues tiene su propio espacio de memoria y conjunto completo y privado de recursos básicos en tiempo de ejecución.

Forman parte de la programación concurrente, donde se pueden ejecutar tareas simultáneas.

3.1.4 java.util.ArrayList

Clase que utiliza una matriz dinámica para el almacén de elementos, los puntos importantes son:

1. Puede contener elementos duplicados.
2. Mantiene el orden de inserción.
3. No está sincronizada, lo que indica que dos hilos pueden manipular el mismo objeto de forma simultánea.
4. Permite acceso aleatorio debido a que la matriz funciona en base al índice.
5. Su manipulación puede verse lenta debido a las operaciones necesarias cuando se eliminan muchos elementos de la lista de matrices.

Su uso e iteración están descritos en el siguiente código:

```
//Inicialización de lista
ArrayList<String> list=new ArrayList<String>();//Creación de ArrayList
list.add("Ravi");//Agregar objetos
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Listado de contenido
Iterator itr=list.iterator();
while(itr.hasNext()){
  System.out.println(itr.next());
}
```

3.1.5 java.util.HashMap

Clase que crea un mapa de relación a una tabla de contenido de valores únicos, su uso principal en este trabajo es la asignación de una llave y un valor, donde cada valor puede ser encontrado sabiendo la llave a la que está relacionado.

Los puntos importantes son:

1. Contiene una relación de valores y llaves.
2. Contiene elementos únicos de llaves y valores.
3. Permite llaves y valores nulos.
4. No guarda ningún orden.
5. Las llaves y valores son de tipo Objeto.

El siguiente código ejemplifica su uso:

```
//Declaración
HashMap<Integer,String> hm=new HashMap<Integer,String>();
//Agregado de elementos Llave-Valor
hm.put(100,"Amit");
hm.put(101,"Vijay");
hm.put(102,"Rahul");
//Recorrido de los elementos.
for(Map.Entry m : hm.entrySet()){
  System.out.println(m.getKey() + " " + m.getValue());
}
```

3.2 .NET C#

Es un lenguaje orientado a objetos seguro de tipo que permite a los desarrolladores crear una variedad de aplicaciones seguras y robustas que se ejecutan en .NET Framework. Se puede utilizar C# para crear aplicaciones cliente de Windows como servicios Web XML, componentes distribuidos, aplicaciones cliente-servidor o aplicaciones de base de datos.

El lenguaje proporciona características de gran alcance tales como tipos de valores anulables, enumeraciones, delegados, expresiones lambda y acceso directo a memoria que no se encuentran en Java. También admite métodos y tipos genéricos que proporcionan mayor seguridad y rendimiento de tipo e iteradores que permiten a los implementadores de clases de colección definir comportamientos de iteración personalizados que son fáciles de usar por código de cliente.

Como un lenguaje orientado a objetos soporta los conceptos de encapsulación, herencia y polimorfismo. Todas las variables y métodos, incluido el método Main, el punto de entrada de la aplicación, se encapsulan dentro de las definiciones de clase. Una clase puede heredar directamente de una clase padre, pero puede implementar cualquier número de interfaces. Los métodos que reemplazan a los métodos virtuales en una clase padre requieren la palabra clave override como una forma de evitar la redefinición accidental.

3.2.1 .NET Framework

Los programas C# se ejecutan en .NET Framework, un componente integral de Windows que incluye un sistema de ejecución virtual llamado Common Language Runtime (CLR) y con conjunto unificado de bibliotecas de clases. El CLR es la implementación comercial por parte de Microsoft de la Common Language Infrastructure (CLI), una forma internacional que es la base para crear ambientes de ejecución y desarrollo en los que los lenguajes y las bibliotecas trabajan juntos sin problemas.

El código fuente escrito en C# se compila en un lenguaje intermedio (IL) que se ajusta a la especificación CLI. El código IL y los recursos, como bitmaps y cadenas, se almacenan en un archivo ejecutable denominado “assembly”, normalmente con una extensión .exe o .dll. Un ensamblado contiene un manifiesto que proporciona información sobre los tipos de ensamblado, la versión, la cultura y los requisitos de seguridad.

Cuando se ejecuta el programa C#, el ensamblado se carga en el CLR, que puede tomar varias acciones basadas en la información del manifiesto. A continuación, si se cumplen los requisitos de seguridad, el CLR realiza la compilación para convertir el código IL en instrucciones nativas de la máquina. El CLR también proporciona otros servicios relacionados con la recolección automática de basura, manejo de excepciones y administración de recursos. Código que es ejecutado por el CLR a veces se conoce como “código administrado”, en contraste con “código no administrado” que se compila en el lenguaje de máquina nativo que se dirige a un sistema específico, El siguiente diagrama ilustra las relaciones de compilación y tiempo de ejecución de los archivos de código fuente de C#, las bibliotecas de clase de .NET Framework, los ensamblados y el CLR.

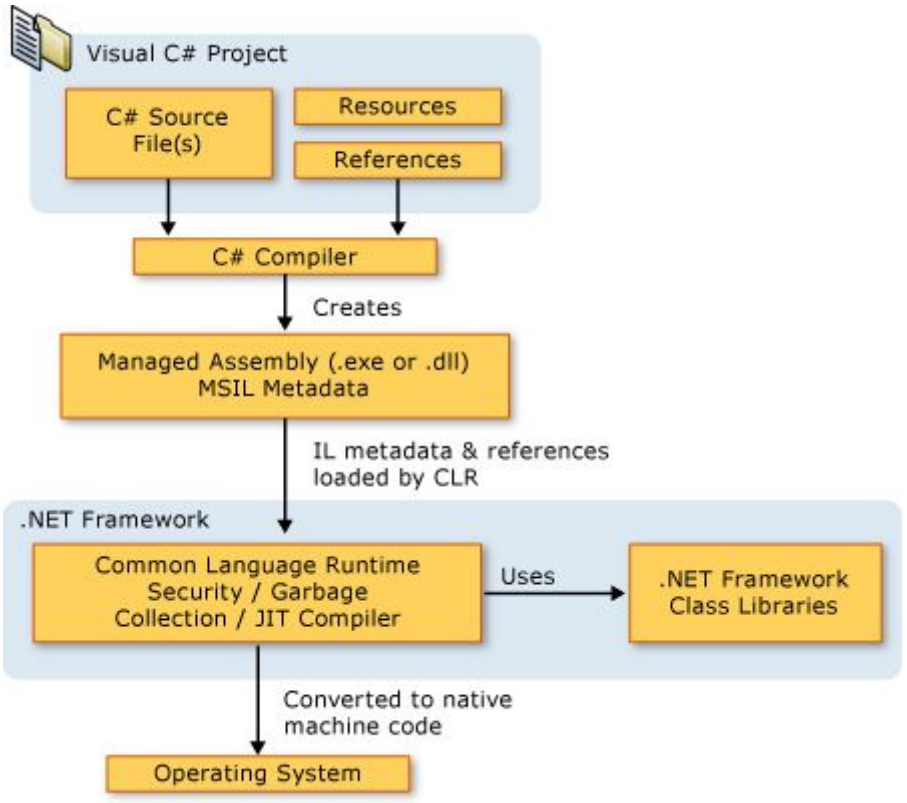

Figura 2. Relación de compilación de código C#

La interoperabilidad del lenguaje es una característica de .NET Framework, dado que el código IL producido por el compilador C# cumple con la especificación de tipo común (CTS), el código IL generado desde C3 puede interactuar con el código generado a partir de las versiones .NET de Visual Basic, Visual C++ o cualquier de los otros 20 lenguajes compatibles con CTS. Un Único ensamblado puede contener múltiples módulos escritos en diferentes lenguajes .NET, y los tipos pueden referirse entre sí como si estuvieran escritos en el mismo idioma.

Además de los servicios de tiempo de ejecución, .NET Framework también incluye una extensa biblioteca de más de 4000 clases organizadas en espacios de nombres que proporcionan una amplia variedad de funcionalidades útiles para todo, desde la entrada y salida de archivos hasta la manipulación de cadenas. La aplicación típica utiliza ampliamente la biblioteca de clases .NET Framework para gestionar las tareas comunes.

3.3 JavaScript

Es un lenguaje de programación orientado a objetos perteneciente a los lenguajes tipo script, su ambiente de ejecución es en navegadores web.

Fué inventado por Brendan Eich en 1995, y se convirtió en un estándar de programación en 1997, ECMAScript es el nombre oficial de este lenguaje.

Los objetos más utilizados en este trabajo son:

3.3.1 Object

Es un elemento que posee propiedades y métodos como parte de su definición, un objeto puede estar vacío.

Ejemplo:

```
var person = {
     firstName:"John",
     lastName:"Doe"
};
```

El objeto “person”, tiene como propiedades “firstName” y “lastName”, además de cada propiedad tiene una asignación de valor “John” y “Doe” respectivamente.

3.3.2 Arrays

Es una matriz que puede contener más de un valor a la vez bajo un mismo nombre y se puede acceder a los valores haciendo referencia a un número de índice.

Definición de una arreglo:

```
var cars = ["Saab", "Volvo", "BMW"];
```

El objeto “cars” tiene una lista de objetos bajo su propio nombre, “Saab”, “Volvo” y “BMW”, son parte de la lista de objetos que le pertenecen.

3.3.3 Funciones

Es un bloque de código diseñado para realizar una tarea en particular, es llamada por otro bloque de código que puede o nó regresar un valor final al bloque de código que solicitó la función.

Ejemplo:

```
function nombre( parametro1, parametro2, parametro3) {
    código que será ejecutado
}
```

3.3.4 AJAX

AJAX (Asynchronous JavaScript And XML), referente a la combinación de un objeto XMLHttpRequest y JavaScript con el fin de solicitar datos a servidores de manera asíncrona, característica que da grandes beneficios en el transporte de información, algunas de ellas son:

- Actualizar datos sin recargar la página
- Solicitar datos de un servidor
- Recibir datos de un servidor
- Enviar datos a un servidor

3.4 Sencha Ext JS

Es un Framework de desarrollo basado en JavaScript para la construcción de aplicaciones web de gran densidad de datos, aprovecha las características de HTML5 en los navegadores modernos.

Cuenta con más de 115 componentes de interfaz de usuario de alto rendimiento, como calendarios, tablas, árboles, gráficos, ventanas y mas. Permite el fácil consumo de cualquier fuente de datos back-end, además contiene componentes avanzados dedicados a la visualización de grandes cantidades de información y su fácil representación en vistas web.

3.4.1 Ext.grid.Panel

Componente en forma de tabla, ideal para la vista de grandes cantidades de datos en un navegador web, permite la fácil búsqueda, ordenado y filtrado de datos.

Este componente está formado por dos piezas principales: **Ext.data.Store** que es el contenedor de datos y un conjunto de columnas para ser procesadas en una vista al usuario.

Ext.data.Store: Es el contenedor que guarda un chaché de lado del cliente en objetos tipo **Ext.data.Model**, estos últimos definen la estructura que tiene la información que será mostrada, donde los datos generalmente son obtenidos a través del objeto **Ext.data.proxy.Proxy**, mismo que facilita la carga y guardado de los datos para su utilización por el componente principal Ex.gid.Panel.

3.4.2 Ext.data.BufferedStore

Es un contenedor que mantiene un mapa de páginas escasamente poblado que corresponde a un conjunto de datos extremadamente grande del lado del servidor.

Cuando se utiliza un BufferedStore no todos los conjuntos de datos están presentes en el cliente. Sólo estarán presentes las páginas que hayan sido solicitadas por la interfaz de usuario (generalmente Ext.grid.Panel) y las páginas circundantes. La retención de páginas vistas en BufferedStore después de haberlas desplazada fuera de la vista es configurable.

Una vez iniciada una instancia de **Ext.grid.plugin.BufferedRenderer**, que es el componente que agrega esta funcionalidad a un Ext.grid.Panel, comienza la supervisión del desplazamiento de la cuadrícula y actualizará las filas de la vista del caché de páginas según sea necesario. También extraerá nuevos datos en el caché de página cuando el desplazamiento de la vista se acerque a los datos de cualquiera de los extremos superiores e inferiores.

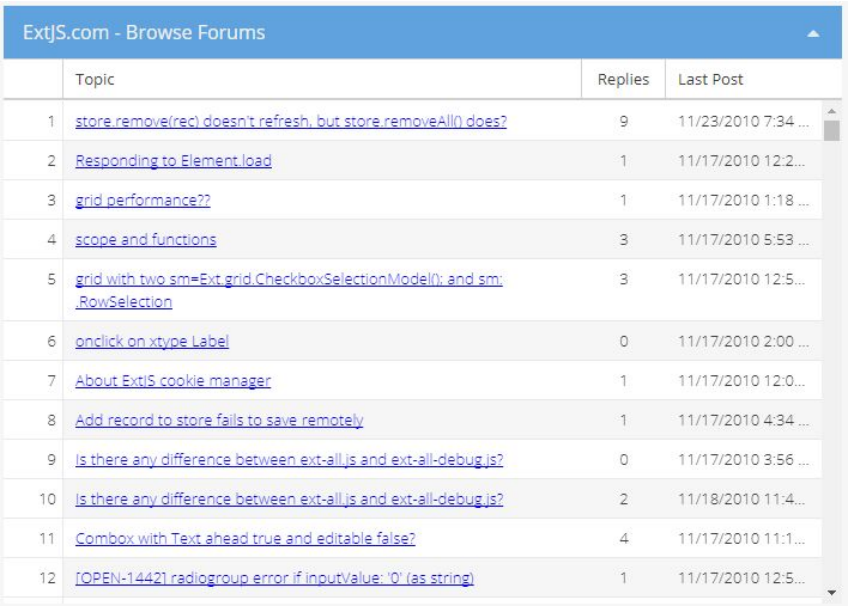

Figura 3. Tabla con búfer de Ext JS

3.4.3 Ext.tree.Panel

Componente que proporciona una representación de datos en estructuras de árbol. El contenedor que lleva los datos de entrada a este componente están en el objeto **Ext.data.TreeStore**. Se pueden configurar múltiples columnas además de la que contiene la estructura arbórea.

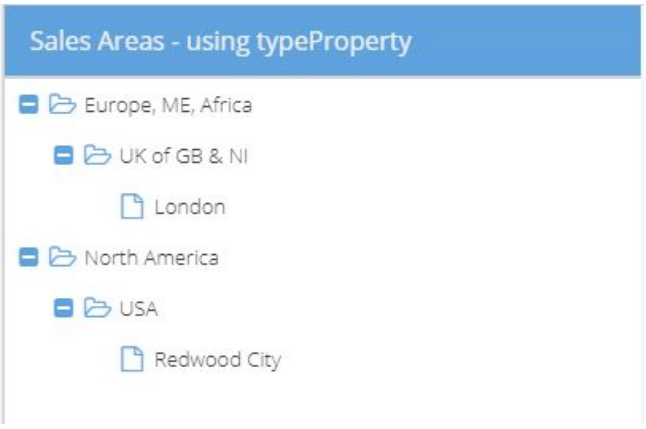

Figura 4. TreePanel de Ext JS

Ext.data.TreeStore: Es el contenedor de datos que posee el nodo raíz de un árbol y proporciona métodos para cargar datos locales o remotos como nodos secundarios de la raíz y cualquier nodo descendiente, dentro de sus funcionalidades está el filtrado de información, cuando un nodo es filtrado el contenedor puede responder con los elementos que cumplen con el patrón de selección haciendo visible solo los nodos que se encuentran bajo la búsqueda.

3.4.4 Ext.window.Window

Es un panel especializado para ser utilizado como una ventana de aplicación. Este tipo de ventanas son flotantes, redimensionables y arrastrables por defecto, se puede maximizar para cubrir la vista del usuario, minimizar y restaurar a su estado anterior.

Este elemento funciona como contenedor pues puede guardar y organizar en su estructura más componentes secundarios que estarán presentes en la vista final.

Figura 5. Vista de una ventaja Ext JS

3.4.5 Ext.data.proxy.Ajax

Es uno de los componentes más utilizados para obtener datos externos. Utiliza solicitudes AJAX para cargar datos de servidores, generalmente para ser almacenados en un objeto Ext.data.Store.

Una de las limitantes de la herramienta es que no puede obtener datos de un dominio diferente pues la política de los navegadores actuales restringe este tipo de solicitudes a dominios externos.

3.6 SQLLoader

Es una herramienta de carga de información a Base de Datos que forma parte de las utilerías del cliente Oracle 12C.

Utiliza datos externos para la alimentación a Bases de Datos Oracle, tiene un poderoso motor de análisis de datos que pone poca limitación en el formato del archivo de datos, se puede utilizar SQLLoader para lo siguiente:

- Carga de información en red a partir de archivos externos a la base de datos.

- Carga de varios archivos en la misma sesión.
- Carga de datos en varias tablas durante la misma sesión.
- Carga selectiva, se puede alimentar información basada en valores de los registros
- Manipulación de datos previa a la carga.
- Generación de informes de errores sofisticados.
- Se pueden cargar archivos de datos secundarios para cargar LOB y colecciones.

Se puede utilizar SQLLoader de dos maneras: con o sin un archivo de control. Un archivo de control maneja el comportamiento de SQLLoader y uno o más archivos de datos utilizados en la carga. El uso de un archivo de control da mayor manejo sobre la operación de carga, lo que podría ser deseable para situaciones de alimentación más complejas. Pero para cargas simples, puede utilizar SQLLoader sin especificar un archivo de control.

La salida de SQLLoader es una base de datos Oracle (donde se cargan los datos), un archivo de registro, un archivo defectuoso si hay registros rechazados y potencialmente un archivo de descarte.

La siguiente figura muestra un ejemplo del flujo de una sesión que utiliza un archivo de control:

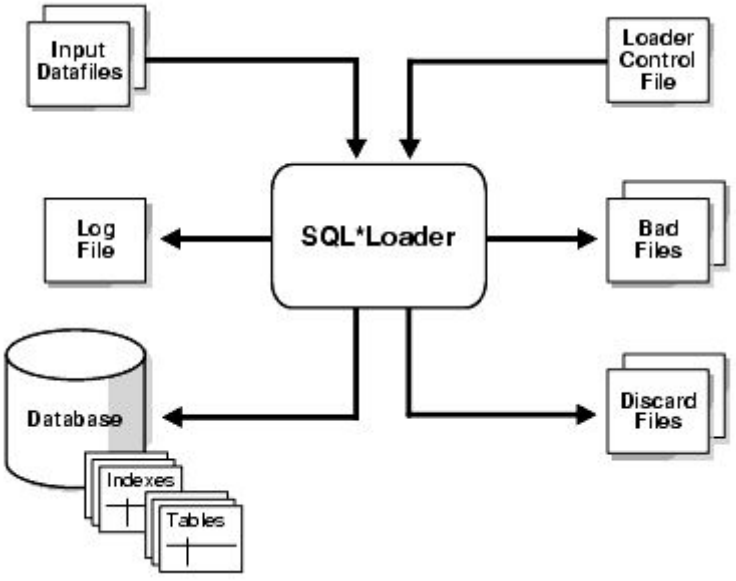

Figura 6. Flujo de carga SQLLoader

3.7 Cubo OLAP

Un cubo OLAP es una estructura de datos que supera las limitaciones de las bases de datos relacionales y proporciona un análisis rápido de datos. Los cubos pueden mostrar y sumar grandes cantidades de datos, a la vez que proporcionan a los usuarios acceso mediante búsqueda a los puntos de datos. De este modo, los datos

se pueden resumir o reorganizar según sea necesario, para procesar la variedad más amplia de preguntas pertinentes al área de interés de un usuario.

Las bases de datos que una empresa utiliza para almacenar sus transacciones y registros se denominan bases de datos de procesamiento de transacciones en línea ***(OLTP)****. Normalmente, estas bases de datos tienen registros que se introducen uno a uno y que contienen una gran cantidad de información, que los estrategas pueden utilizar para tomar decisiones fundamentadas sobre sus negocios. Sin embargo, las bases de datos que se utilizan para almacenar los datos no se diseñaron para el análisis. Por lo tanto, obtener respuestas de estas bases de datos requiere tiempo y esfuerzo. Las bases de datos OLAP son bases de datos especializadas, diseñadas para ayudar a extraer esta información de inteligencia empresarial de los datos.*

Origen de datos: *Un origen de datos es el origen de todos los datos contenidos dentro de un cubo OLAP. Un cubo OLAP se conecta a un origen de datos para leer y procesar los datos sin procesar para llevar a cabo cálculos y agregaciones para sus medidas asociadas. El origen de datos de todos los cubos OLAP se denominan* ***Data Marts****.*

Dimensiones: *Una dimensión hace referencia a una extensión del almacenamiento de datos. Las dimensiones permiten el filtrado, la agrupación y el etiquetado de datos. Por ejemplo, puede filtrar los equipos por el sistema operativo instalado y agrupar a los usuarios en categorías por sexo o edad. A continuación, los datos se presentan en un formato donde se clasifican de forma natural en estas jerarquías y categorías para permitir un análisis más exhaustivo. Las dimensiones también pueden tener jerarquías naturales para permitir a los usuarios "profundizar" hasta niveles más precisos de detalle. Por ejemplo, la dimensión Fecha tiene una jerarquía de la que se pueden obtener más detalles por Año, a continuación Trimestre, a continuación, Mes, a continuación, Semana y a continuación Día.*
La ilustración siguiente muestra un cubo OLAP que contiene las dimensiones Fecha, Región y Producto.

Figura 7. Cubo OLAP

Obtención de detalle: *Cuando un usuario profundiza en los datos en un cubo OLAP para obtener detalles, está analizando los datos a otro nivel de resumen. El nivel de detalle de los datos cambia en función de la obtención de detalle aplicada por el usuario para examinar los datos a distintos niveles en la jerarquía. A medida que el usuario aumenta el nivel de obtención de detalle, pasa de la información de resumen*

a los datos con un enfoque más reducido. Los siguientes son ejemplos de obtención de detalles:

- *Obtención de detalles de la información demográfica sobre la población de EE.UU. y, a continuación, del Estado de Washington y, a continuación, del área metropolitana de Seattle y, a continuación, de la ciudad de Redmond y, finalmente, de la población de Microsoft.*
- *Obtención de detalles de las cifras de ventas de las consolas Xbox 360 en 2011 y, a continuación, en el cuarto trimestre del año y, a continuación, en el mes de diciembre y, a continuación, en la semana previa a Navidad y, finalmente, en Nochebuena.*

Perforación: *Cuando los usuarios "perforan" los datos desean ver todas las transacciones individuales que han contribuido a los datos agregados del cubo OLAP. En otras palabras, el usuario puede recuperar los datos con un nivel de detalle menor para un valor de medida determinado. Por ejemplo, si tiene los datos de ventas de un mes y una categoría de producto concretos, puede perforar dichos datos para ver una lista de cada fila de la tabla contenida dentro de esa celda de datos.*

Es fácil confundir los términos "obtención de datos" y "perforación". La diferencia principal entre ellos es que una obtención de datos funciona en una jerarquía predefinida de datos: por ejemplo, EE.UU y, a continuación, Washington y, a continuación, Seattle, dentro del cubo OLAP. Una perforación va directamente al nivel más pequeño de detalle de datos y recupera un conjunto de filas del origen de datos que se ha agregado en una sola celda.

Agregaciones: *Las agregaciones de un cubo OLAP son conjuntos de datos con resúmenes previos. Son análogos a una instrucción SELECT de SQL con una cláusula GROUP BY.* [5]

[5] *https://msdn.microsoft.com/es-es/library/hh916536(v=sc.12).aspx*

4 Antecedentes del proyecto

Las áreas generadoras de información trabajan en gran medida con hojas de cálculo para realizar formulaciones y estimaciones económicas, que finalmente pasarán al área de sistematización para que los datos sean tratados de forma automática y no de forma manual como se hace en un principio con este tipo de herramientas.

4.2 Carga masiva de información

La alimentación de bases de datos es un procedimiento elemental en la Institución, la rapidez y fiabilidad en este proceso es indispensable pues la magnitud de la información en general es muy grande.

4.2.1 Requerimiento

Alimentar los datos extraídos de los archivos Excel a una base de datos Oracle 12g para su tratamiento en departamento de Sistematización.

4.2.2 Estructuración

Una vez extraídos los datos pasa por un proceso estructuración y barrido que permite relacionar los datos para darles un significado arbóreo. Las estructuras en árbol se encuentran en el eje vertical (bloque rojo) y el eje horizontal (bloque verde).

Se estructuran para que cada elemento que tenga un superior, sea marcado con el ID que le corresponde al elemento superior, así se podrá reconstruir la estructura sin perder la relación original.

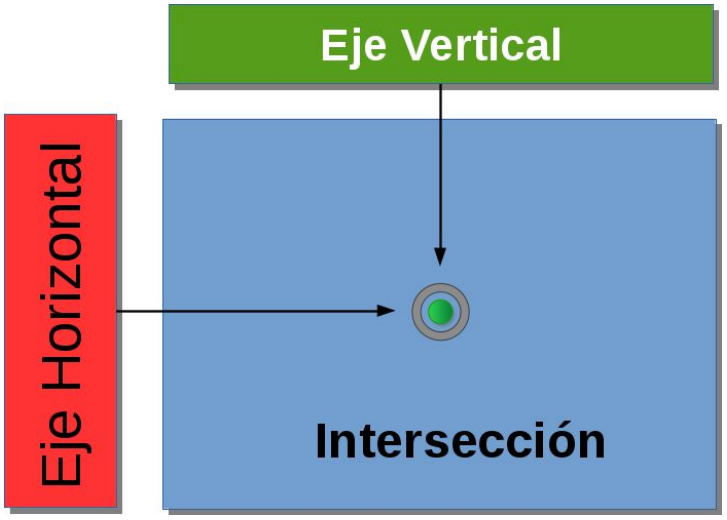

Figura 8. Cruce de datos

4.2.3 Alimentación

Una vez extraídos y estructurados los datos es necesario sean alimentados a la base de datos.

El proceso con el que se alimenta es mediante instrucciones INSERT que son generadas como cadena para ser ejecutadas directamente mediante el conector JAVA que se tiene implementado.

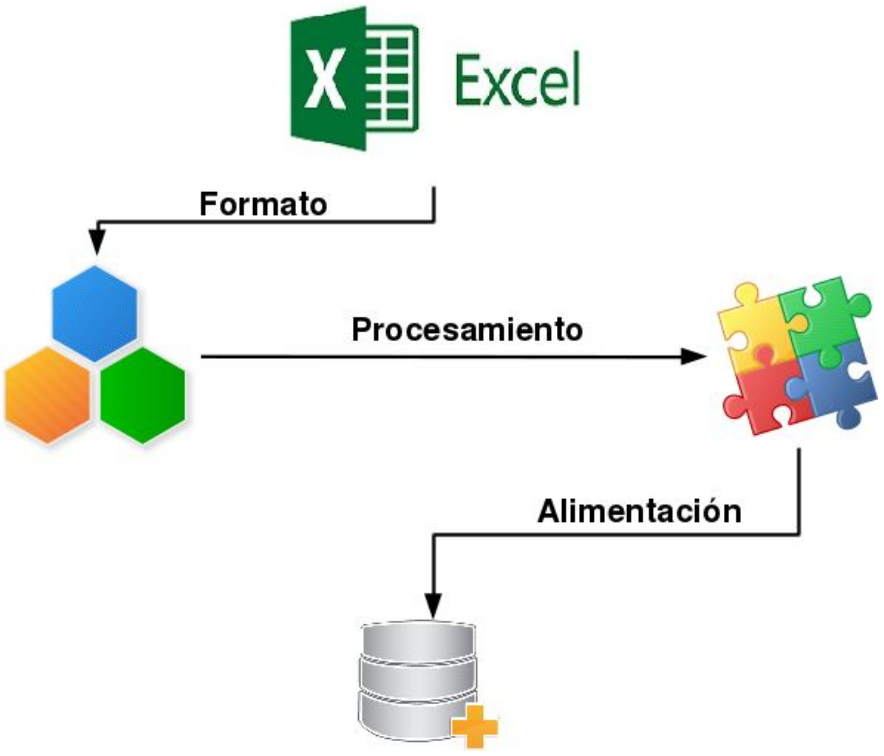

Figura 9. Flujo de alimentación a partir de archivo Excel

4.2.4 Problemática

La alimentación de datos por inserciones de este tipo es muy lenta para cantidades grandes, de más de 10,000 instrucciones, donde actualmente se habla de cientos de miles o millones de datos por carga, así que este método de alimentación deja de ser una buena práctica para las necesidades que se tienen.

5 Análisis y metodología empleada

De acuerdo al tipo de proyectos manejados en el Instituto me enfoque en la metodología Modelo, Vista, Controlador (MVC)[6].

Utilicé esta arquitectura por su modularidad y por su capacidad de mantenimiento por parte de diferentes equipos de desarrollo, además que establece la separación de los elementos modulares de una aplicación para usuario final como era el caso de los proyectos presentados por este trabajo.

Modelo: Cada aplicación desarrollada en el Instituto está sustentada por la información existente en bases de datos, las actividades más básicas extraen información de estos contenedores, generalmente por las continuas modificaciones que la información soporta. Se requiere que en todo momento el personal que explota los datos que son mostrados tenga una pantalla actualizada, con la totalidad de registros y con la mayor precisión posible, esto por la cantidad de verificaciones y comprobaciones que se realizan sobre los datos para determinar su veracidad además de su fiabilidad.

Vista: Las necesidades persistentes en el Instituto son la presentación de datos para usuario final, que requiere observar un conjunto de datos en su mayor cantidad y calidad posible, requiriendo la generación de gráficos además de componentes que puedan expresar con la mayor claridad posible la estructura, relación y precisión, pues estos elementos en los datos, tienen también significado además de su valor, principalmente en la estructura.

Los insumos a las aplicaciones son valores brutos que es necesario transformar en pantallas enriquecidas con elementos filtrados, gráficos ó mediante búsquedas que puedan puntualizar la información que se observa, además de presentarla de la manera más comprensible

Controlador: El tratamiento de la información antes y después de ser mostrada al usuario es una operación recurrente en el uso de las aplicaciones desarrolladas, es necesario solicitar datos a diferentes fuentes de información, incluyendo las que el usuario final pueda brindar, generar un procesamiento de los datos recibidos y crear nuevamente una salida que visualice los cambios realizados.

La cantidad de información empleada en los programas es la causa de dedicar mayores recursos a su tratamiento, pues se deben generar procedimientos que puedan procesar la mayor cantidad de datos posibles con la menor cantidad de recursos posibles, esto ha sido posible con el diseño de estructuras optimizadas y uso de técnicas que solventan el paso de información de gran tamaño.

[6] https://desarrolloweb.com/articulos/que-es-mvc.html

En cuanto a la vida de un proyecto, utilicé las siguientes etapas:

- **Requerimientos:** Es la etapa en la que se investiga el conjunto de características y especificaciones que tiene una necesidad o problemática.
- **Solución:** Es la etapa en la que se genera una solución o soluciones para los requerimientos entregados en la etapa anterior.
- **Diseño:** Etapa en la que se inicia un proceso de diseño que solvente las necesidades a partir de las soluciones propuestas por la etapa anterior.
- **Desarrollo:** En esta etapa se comienza implementación del diseño generado por la etapa anterior, además de la inclusión de herramientas que puedan solventar el diseño en cuanto a funcionalidad y presentación.
- **Pruebas:** Etapa en la que se crea un conjunto de ejercicios de prueba para la obtención de resultados deseados, así como la generación de informes respondiendo a la satisfacción o no, de los requerimientos.
- **Producción:** Etapa de lanzamiento en ambientes dedicados a soportar las necesidades de la aplicación final, lista para ser utilizada por los usuarios.

Controlando la vida de un proyecto bajo estas etapas me fue posible llevar los proyectos aquí descritos minimizando problemáticas en el desarrollo de las herramientas, principalmente la demora por el regreso a etapas anteriores que deban ser ajustadas o modificadas.

6 Participación profesional

A continuación se describen los desarrollos e implementaciones que desarrolle para el Instituto, donde cada proyecto está dirigido a cumplir con los siguientes objetivos:

- Sistematización
- Eficiencia.
- Confidencialidad.
- Integridad.
- Disponibilidad.

6.2 Envío de archivos para carga masiva de información

Para cumplir con este requerimiento implementé y desarrollé los medios de comunicación para realizar transferencias de archivos a través de tres de los métodos más utilizados (FTP, SSH y SOCKET).

Para los métodos FTP y SSH, implemente solo el extremo “cliente” pues el servidor Oracle Solaris con el que se cuenta ya tiene habilitados estos servicios de comunicación. En cuanto a SOCKET si fue necesario el desarrollo de la aplicación “cliente” y la aplicación “servidor”.

6.2.1 Lenguaje de programación

Seleccioné Java por su alta compatibilidad con las nuevas tecnologías, además del gran soporte de herramientas desarrolladas por comunidades de software libre.

6.2.2 Protocolos de transferencia

Por motivos de confiabilidad de carga y en favor de abarcar el conjunto de servicios de transferencia ya disponibles en el Servidor de Base de Datos, implementé los siguientes métodos de transferencia:

- FTP
- SSH
- SOCKET

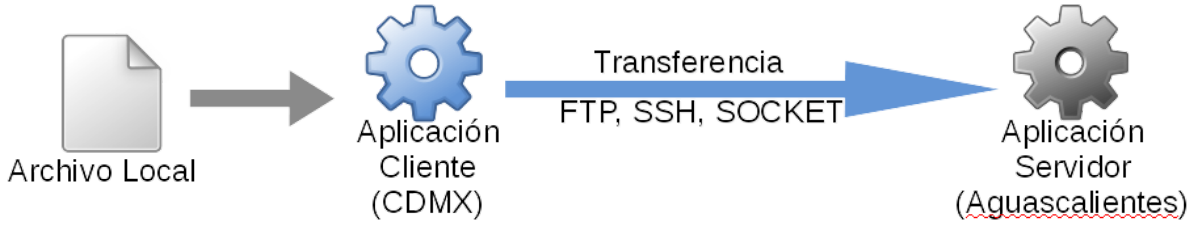

Figura 10. Transferencia de archivos planos

6.2.2.1 FTP Cliente

Implementé la conexión y transferencia de archivos utilizando la biblioteca "**org.apache.commons.net.ftp.FTPClient**" que permite realizar la autenticación y el envío de flujo de bytes por conexión TCP, este servicio ya existe por parte del servidor de base de datos, implementado con la herramienta nativa FTPSERVER.

El establecimiento de la conexión se logra a través de cuatro parámetros indispensables:

1. IP del servidor (**host**)
2. Puerto de servicio (**port**)
3. Usuario (**user**)
4. Contraseña (**pass**)

A continuación se muestra el código que implementa la conexión al servidor de FTP.

```
public boolean connect() throws Exception {
    if (!ftpClient.isConnected()) {
        try {
            //Solicitud de conexión
            ftpClient.connect(host, port);
            ftpClient.setBufferSize(65536);
            ftpClient.enterLocalPassiveMode();
            //Autenticación de usuario
            return ftpClient.login(user, pass);
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    } else {
        return false;
    }
}
```

La transferencia de datos, se realiza leyendo el flujo de entrada, perteneciente al archivo local, por cada lectura de bloques se realiza al mismo tiempo una escritura de la misma magnitud al flujo de salida, el código que realiza esta operación, es el siguiente:

```
//lectura de flujo de entrada
while ((bytesRead = inputStream.read(buffer)) != -1) {
    //escritura a flujo de salida
    outputStream.write(buffer, 0, bytesRead);
    //conteo de datos leídos
    totalBytesRead += bytesRead;
    //registro de porcentaje completado
    percentCompleted = (int) (totalBytesRead * 100 / localFileSize);
    //reporte de progreso
    communicationManager.updateCurrentMessage("[FTP] Transferencia", "Archivo [" +
localFile.getName() + "] destino [" + remote + "].");
    communicationManager.updateCurrentProgress("[FTP] Transferencia",
percentCompleted);
    //condición de salida, para lectura y escritura de flujos
    if (totalBytesRead == localFileSize) {
        break;
    }
}
```

En la implementación de FTP, agregue la funcionalidad de reintento de transferencia, esto se logra enviando solo la porción de datos que faltan por enviar y habilitando por ejemplo en ***VSFTPD*** la opción de ***delete_failed_uploads*** en falso, esto es por evitar que el servidor FTP, deseche los archivos que fueron interrumpidos, esta funcionalidad es aprovechada por el siguiente código para terminar de enviar el archivo, esto en lugar de enviarlo por completo en los siguientes reintentos:

```
//valida que sea posible una continuación de datos
if (continueTransfer && totalBytesRead != 0) {
    buffer = new byte[bufferSize];
    //cálculo de datos que deben ser ignorados en la lectura del archivo local
    long skipBufferCount = bufferSize < totalBytesRead ? totalBytesRead / bufferSize : 1;
    buffer = new byte[bufferSize];
    //procedimiento de salto de bytes
    for (long currentByte = 0; currentByte < skipBufferCount; currentByte++) {
        inputStream.read(buffer);
    }
    //leer restante si aun falta por leer una fracción de búfer.
    if (bufferSize * skipBufferCount < totalBytesRead) {
        inputStream.read(new byte[(int) totalBytesRead - ((int) bufferSize * (int)
skipBufferCount)]);
    }
} else {
    //solicita su eliminación en caso de que ya exista un archivo con el mismo nombre
    ftpClient.deleteFile(remote);
}
```

El código anteriormente descrito es de gran ayuda, sobre todo para archivos de gran tamaño que son enviados por un canal de baja velocidad y confiabilidad, como es el caso

de los envíos de datos entre Ciudad de México y Aguascalientes, así es posible culminar una tarea de transferencia de datos sin necesidad de comenzar de nuevo cada vez.

6.2.2.2 SSH Cliente

Realicé la implementación de conexión y transferencia de archivos utilizando la biblioteca "**com.jcraft.jsch.JSch**" que permite realizar la autenticación y el envío de flujo de bytes por conexión TCP, este servicio ya existe por parte del servidor de base de datos, implementado con la herramienta nativa **sshd**.

Los parámetros requeridos para este tipo de conexión segura, son cuatro:

1. IP del servidor (**host**)
2. Puerto de servicio (**port**)
3. Usuario (**user**)
4. Contraseña (**pass**)

El código correspondiente a esta operación es el siguiente:

```
try {
    //Inicialización de objeto
    jsch = new JSch();
    session = jsch.getSession(user, host, port);
    //se especifica el tipo de autenticación que se realizará
    session.setConfig("PreferredAuthentications", "password");
    //Autenticación de usuario
    session.setPassword(pass);
    UserInfo ui = new MyUserInfo();
    session.setUserInfo(ui);
    //Solicitud de conexión
    session.connect();
    return true;
} catch (Exception e) {
    e.printStackTrace();
    return false;
}
```

Agregué un procedimiento para poder transferir y reanudar una transferencia que pudo haber sido interrumpida, donde la principal ventaja es que no es necesaria la retransmisión completa de los datos, sino solo de los que faltan.

```
long localFileSize = localFile.length();
//flujo de entrada para archivo local
fis = new FileInputStream(localFile);

Channel channel = session.openChannel("sftp");
channel.connect();
channelSftp = (ChannelSftp) channel;
```

```
//Obtención de cantidad de datos transmitidos
String totalBytesResponse = continueTransfer ? execResponse("wc -c " + remote + "|awk
'{print $1}'", "Obtiene tamaño de archivo.").toString() : "";
long totalBytesRead = totalBytesResponse.equals("") ? 0 :
Integer.parseInt(totalBytesResponse.trim());

//Creación de hilo encargado del monitoreo de la transferencia
SftpProgressMonitor monitor = new ProgressMonitor(totalBytesRead, localFileSize);

//Transferencia de archivo ó continuación de transferencia de archivo
channelSftp.put(fis, remote, monitor, continueTransfer ? ChannelSftp.RESUME :
ChannelSftp.OVERWRITE);
```

6.2.2.3 Socket

El siguiente diagrama muestra la interacción que existe entre el servidor de base de datos y las aplicaciones cliente.

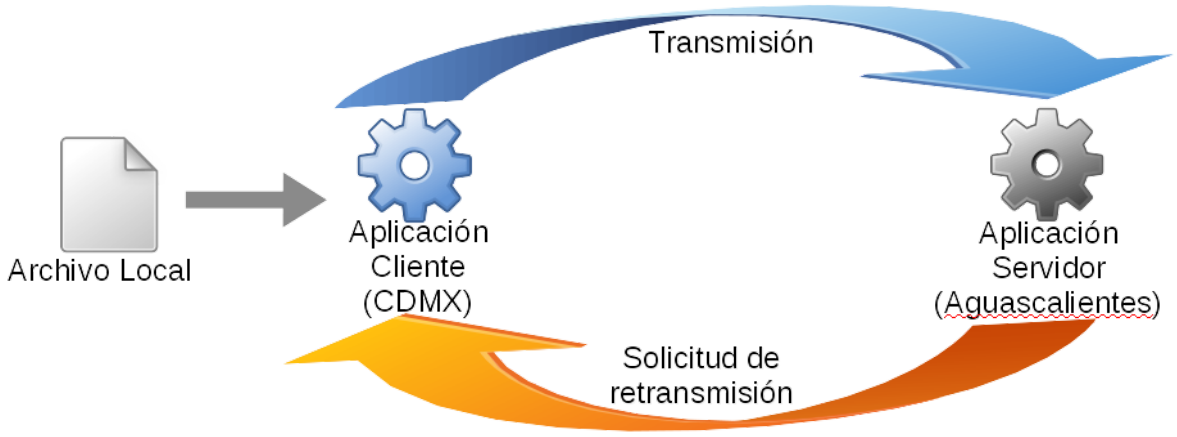

Figura 11. Transferencia de archivos por Socket

6.2.2.3.1 Servidor

Desarrollado en Java bajo la biblioteca nativa de JDK **java.net** utilizando la clase principal, **ServerSocket** para abrir el puerto de comunicación.

Multisesión: El Servidor va a permitir la conexión de más de un cliente a la vez y ser atendidas las peticiones por un hilo de ejecución que recibirá el flujo de bytes y depositarlo en el sistema de archivos local.

```
//Inicialización de Servidor con puerto de escucha InterOpConfig.PORT
ServerSocket serverSocket = new
ServerSocket(Integer.parseInt(InterOpConfig.PORT.getParam()));
```

```
//ciclo infinito para la recepción indeterminada de conexiones
while (true) {
    try {
        //Instrucción de recepción de conexiones
        Socket clientSocket = serverSocket.accept();

        //Creación de hilo encargado de la atención de conexiones entrantes
        InterOPUnitService clientSession = new InterOPUnitService(id, clientSocket, routes,
killedChannels, PPID, nestEnviromentManager, communicationManager);
        clientSession.start();
        id++;
    } catch (Exception e) {
        System.out.println("Error client session[" + e.getMessage() + "]");
    }
}
```

Reintento de conexión: se habilitará el servidor para volver a conectarse en caso de interrupción por parte de red o de error de escritura.

```
//Instrucción inicial de recepción de archivo
String responseReceiver = fileReceiver(in, out, parameters);
if (responseReceiver.equals("OK!")) {
    out.writeUTF("Proceso de transferencia terminado");
    while (true) {
        responseReceiver = in.readUTF();
        //Valida la petición del cliente, en caso de que requiera una continuación de envío
        if (responseReceiver.startsWith("FILE")) {
            parameters.clear();
            parameters.put(responseReceiver.split(":")[0], responseReceiver.split(":")[1]);
            receiveParameters(parameters);
            out.writeUTF("Proceso de envío iniciado");

            //Inicia nuevamente procedimiento de recepción de datos
            responseReceiver = fileReceiver(in, out, parameters);
            if (responseReceiver.equals("OK!")) {
                out.writeUTF("Proceso de transferencia terminado");
            } else {
                out.writeUTF("Proceso de transferencia fallido, Msg: " + responseReceiver);
            }
        } else {
            //Finaliza ciclo de decisiones para dar por concluida la transmisión
            break;
        }
    }
```

Continuación de transferencia: se habilitará el servidor para enviar las partes faltantes de un archivo en caso de interrupción de transferencia.

La lógica que rige esta funcionalidad está descrita en el siguiente código:

```
//Solicitud de tamaño del archivo remoto
long remoteFileSize = in.readLong();
long totalBytesRead = 0;

//Obtención de tamaño del archivo local
long localFileSize = localFile.length();

//Se cuestiona al cliente para saber si requiere una continuación de transferencia de datos
if (in.readBoolean()) {

    //En caso afirmativo se comunica al cliente cual es la porción de datos con la que ya se
cuenta
    out.writeLong(localFileSize);

    //Se verifica que aun falten datos por ser entregados
    if (localFileSize < remoteFileSize) {
        totalBytesRead = localFileSize;
    } else {
        receiveBytes = false;
    }
}
```

Recepción de datos con compresión: se habilita el servidor para recibir datos de manera comprimida por ZIP, además de aceptar que se soliciten envíos parciales de datos, esta operación es la más importante en el procedimiento de recepción de archivos, para esto se utiliza la biblioteca ***java.util.zip.ZipInputStream***, está encargada de comprimir el flujo de bytes que pasa por el Stream principal, de esta forma se recibe el conjunto de bytes que no son directamente el archivo original, sino que pertenece a los bytes resultantes de una compresión ZIP.

El código resultante de esta operación es el siguiente:

```
//Se inicializa el flujo de Bytes correspondiente al archivo de salida
//se indica además si es un envío parcial o total
//esto se realiza por medio del segundo parámetro de FileOutputStream
bos = new BufferedOutputStream(new FileOutputStream(localFile, (localFileSize <
remoteFileSize)));

//ZipInputStream se encarga de recibir el flujo de bytes provenientes del cliente
//este flujo corresponde al archivo con compresión, por lo que los datos que se van
//a recibir no son directamente el archivo original sino los tratados por ZIP
zipInputStream = new ZipInputStream(in);
zipInputStream.getNextEntry();
```

```
//Lectura de flujo de Bytes con compresión
while ((count = zipInputStream.read(bytes)) > 0) {

    //escritura de archivo de salida
    bos.write(bytes, 0, count);

    //conteo de datos recibidos
    totalBytesRead += count;

    //condición de salida para término de transferencia
    if (totalBytesRead == remoteFileSize) {
        zipInputStream.closeEntry();
        break;
    }
}
```

6.2.2.3.2 Cliente

Desarrollado en Java bajo la biblioteca nativa de JDK (**java.net.Socket**) utilizando la clase principal, Socket para conectar con aplicación Servidor.

Encargada de establecer comunicación, requiere dos parámetros indispensables

1. IP del servidor (**host**)
2. Puerto de servicio (**port**)

```
try {
    //Inicialización del objeto de conexión
    sc = new Socket(host, port);
    //Asignación de flujos de entrada y salida
    in = new DataInputStream(sc.getInputStream());
    out = new DataOutputStream(sc.getOutputStream());
    return true;
} catch (Exception e) {
    e.printStackTrace();
    return false;
}
```

Reintento de conexión: se habilitará el servidor para volver a conectarse en caso de interrupción por parte de red o de error de escritura.

```
//Ciclo de reintentos de conexión
while (!isSend) {
    if (disconnect()) {
        if (connect()) {
```

```
            //Envío de archivo
            isSend = send(local, remote, true);
        } else {
            retries++;
        }
    } else {
        retries++;
    }
    if (retries >= maxRetries) {
        return false;
    }
}
```

Continuación de transferencia: se habilitará el servidor para enviar las partes faltantes de un archivo en caso de interrupción de transferencia.

```
//Envío de parámetro indicador de envío parcial
out.writeBoolean(continueTransfer);
if (continueTransfer) {
    //Recepción de tamaño de archivo remoto, esto indica que cantidad ya ha sido entregada
    remoteFileSize = in.readLong();

    //Validación para salto de Bytes
    if (remoteFileSize < localFileSize) {

        //Indica al flujo de entrada que debe ser ignorada la cantidad de Bytes al inicio de la lectura
        is.skip(remoteFileSize);
        totalBytesRead = remoteFileSize;
    } else {
        sendBytes = false;
    }
}
```

De forma contraria al servidor, se agregó la funcionalidad de comprimir los datos antes de ser enviados al servidor, una vez con compresión se realiza una transformación a flujo de bytes. Así es como incrementé en gran medida la velocidad de las transferencias en el mismo tiempo que se asegura la llegada de la totalidad de los datos.

El código siguiente describe la forma de realizar la compresión:

```
//Inicialización de ZipOutputStream, encargado de la compresión del flujo
zipOutputStream = new ZipOutputStream(out);
```

```
//Parametrización de archivo a comprimir
zipOutputStream.putNextEntry(new ZipEntry(localFile.getName()));

while ((bytesRead = is.read(buffer)) > 0) {

    //Escritura de datos a flujo de salida
    zipOutputStream.write(buffer, 0, bytesRead);

    //Registro de datos leídos
    totalBytesRead += bytesRead;

    //Registro de datos leídos para cálculo de velocidad
    bytesReadForSpeed += bytesRead;

    //Registro de porcentaje de envío
    percentCompleted = (int) (totalBytesRead * 100 / localFileSize);

    //Condición de salida para archivo completado
    if (totalBytesRead == localFileSize) {
        zipOutputStream.closeEntry();
        break;
    }
    //calculo de velocidad
    if (speedInSeconds == 0) {
        speed.setCheckpoint("speed-begin");
        speedInSeconds = 1;
    } else {
        speed.setCheckpoint("speed-end");
        speedInSeconds = speed.getComparationInCheckpoints("speed-begin",
"speed-end");

        //Actualización de tasa de transferencia
        if (speedInSeconds >= 1) {
            //Reporte de transferencia
            System.out.println("Velocidad: [" + getFormatFileSize(bytesReadForSpeed) + "/s"
+ "], Tiempo estimado: [" + getTimeFormat(((localFileSize - totalBytesRead) /
bytesReadForSpeed)) + "], Porcentaje: [" + percentCompleted + " %]");
            speedInSeconds = 0;
            bytesReadForSpeed = 0;
        }
    }
}
```

6.3 Carga masiva de información

Uno de los procedimientos de mayor demanda en el Instituto es la alimentación de información a base de datos de manera eficiente y segura, por ello utilicé SqlLoader como herramienta de carga de información, en primer lugar por realizar cargas con muy altas velocidades y en segundo lugar por la simplicidad que ofrece al conectarse a bases de datos Oracle.

6.3.1 Herramienta de carga de datos

SqlLoader: es la herramienta incluida como parte del cliente de oracle para alimentación de datos a cualquier esquema o usuario en cualquier Servidor al que se tenga acceso por red.

6.3.2 Archivo de configuración

CTL: es el archivo de configuración en texto plano que permite parametrizar y personalizar la carga de datos, la configuración utilizada es la siguiente:

```
//Elementos que describen la capacidad de carga, asignando tamaños de lectura y número de errores permitidos:
OPTIONS
(PARALLEL=true,BINDSIZE=1073741824,READSIZE=1073741824,ROWS=15000,ERRORS=49)

//Instrucción de carga de información:
load data

//Asignación de nombre de archivo para ser leído y cadena de fin de línea:
infile 'Archivo.sql' "str '■'"

//Modo de inserción a la tabla, se indica que los datos sólo serán agregados no reemplazados, en caso de que ya exista el valor que se va a asignar, no se reemplaza, se mantiene el original:
append into table TABLA

//Indicación de los campos que separan a las columnas en el archivo de datos y el carácter de inicio y fin de cadena:
fields terminated by ',' optionally enclosed by "┼"

//Evita la producción de errores al permitir la existencia de columnas con valores nulos:
TRAILING NULLCOLS
```

```
//Nombrado de las columnas tal como existen en base de datos:
(C1,C2,C3,C4,C5,C6,C7,C8,C9 CHAR(32000),C10 CHAR(32000))
```

Decidí tomar valores de caracteres difíciles de encontrar en un texto común, debido a que gran parte de la información que se alimenta es de tipo texto y en muchos casos existen valores que pueden romper el significado del archivo de datos, un ejemplo es el uso del símbolo **'┼'**, en una práctica común se utiliza el apóstrofo para delimitar el inicio y fin de una cadena de texto, como lo sería el ejemplo ***'Esto es una cadena de texto'***, si el texto por su significado contiene un apóstrofo generaría un error en el formato de los datos, pues se incrementa el número de columnas que aparentemente tiene el archivo fuente, de esta manera se evita la aparición de este tipo de errores.

6.3.3 Archivo de datos

SQL: Es el archivo de valores que contiene en texto plano el conjunto de datos que serán alimentados a la base de datos, por motivos de codificación y uso de caracteres especiales, utilice un separador de elementos especial, que permite delimitar donde comienza un dato y donde termina. También existe para delimitar el fin de una línea y evitar confundir los datos de un registro con el siguiente.

```
30,202,1,,1,101,1,1,┼Cadena 1┼,┼Cadena 2┼■
30,202,2,1,3,1,2007,2007,┼Cadena '3'┼,┼Cadena "4"┼■
30,202,3,1,3,1,2008,2008,┼Cadena (5)┼,┼Cadena &6┼■
30,202,4,1,3,1,2009,2009,┼Cadena #7┼,┼Cadena 8%┼■
30,202,5,1,3,1,2010,2010,┼Cadena /9┼,┼Cadena = 10┼■
30,202,6,1,3,1,2011,2011,┼Cadena |11┼,┼Cadena 12°┼■
30,202,7,1,3,1,2012,2012,┼Cadena -13┼,┼Cadena [14]┼■
30,202,8,1,3,1,2013,2013,┼Cadena {15}┼,┼Cadena ¿16?┼■
...
```

Como se observa el formato del archivo de valores en los datos que contienen un elemento de tipo texto, se encierra con el símbolo **┼,** así se evita la confusión con alguno de sus caracteres por parte de la herramienta de alimentación utilizada. De igual manera sucede con el carácter de fin de línea, en lugar de utilizar el convencional que debería ser el carácter **'\n'**, se utiliza un símbolo difícil de encontrar en un texto, el seleccionado es **'■'.**

6.3.4 Ejecución de herramienta de carga de datos

SqlLoader es una herramienta que se ejecuta bajo terminal pues no cuenta con interfaz gráfica para interacción con usuarios.

Hice la ejecución de forma local para eliminar el tiempo de transferencia de datos además de aprovechar que el archivo ya existe en el sistema de directorios local, así la referencia al archivo será con una dirección relativa a la llamada de SqlLoader.

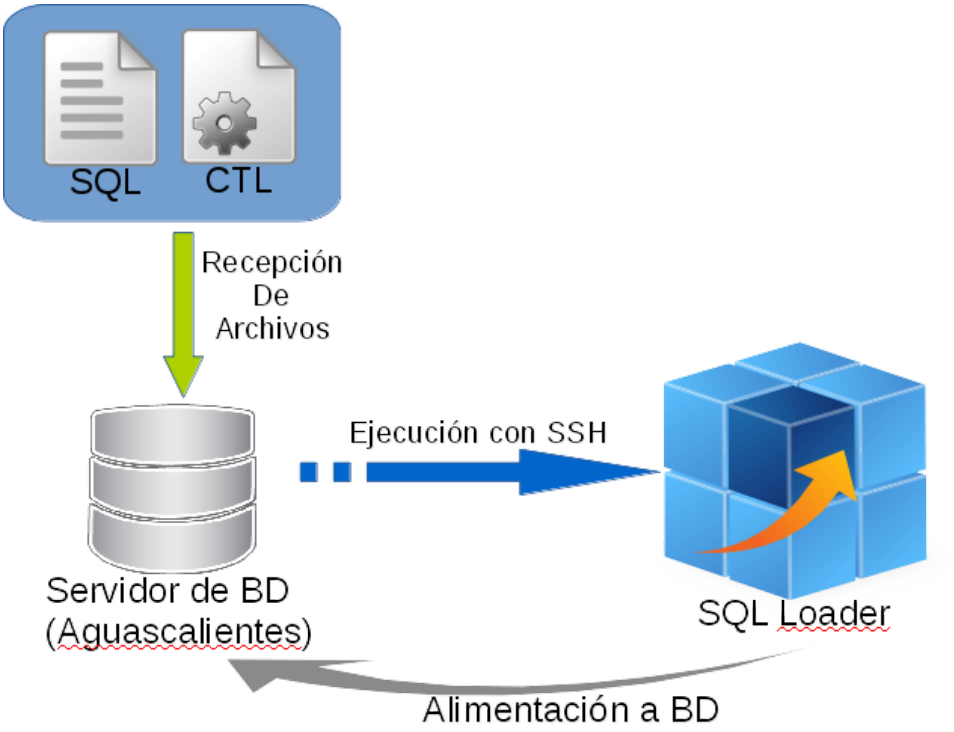

Figura 12. Alimentación mediante SQLLoader

```
sqlldr usuario/contraseña@servidor_oracle/instancia Archivo.ctl
```

Para la sistematización de este procedimiento creé un método en Java nombrado **sendAndLoad,** encargado de toda la interacción con el servidor de Base de Datos, las tareas que debe realizar son las siguientes:

6.3.4.1 Validación de servicio SSH

En este punto se comprueba si existe el servicio SSH, con el fin de garantizar que se pueda ejecutar la orden sqlloader, en caso contrario no tiene sentido continuar con la ejecución, el código que corresponde a este procedimiento es el siguiente:

```
//Inicialización de conexión con parametría: Usuario, Contraseña, Servidor y Puerto
sshLinker = new SshLinker(c.decode(Parameters.SYSUSR.getParam(),
Parameters.KEY.getParam()), c.decode(Parameters.SYSPASS.getParam(),
Parameters.KEY.getParam()), System.getProperty("EVR_INTER_OP_ORACLE_HOST"),
Integer.parseInt(System.getProperty("EVR_INTER_OP_ORACLE_PORT_SSH")),
communicationManager);
```

```
int reConnectionCount = 0;


//Ciclo de reconexión
while (!sshLinker.connect()) {
    sshLinker.disconnect();

    //Instrucción de espera para siguiente conexión
    Thread.sleep(2000);
}
```

6.3.4.2 Validación de servicio Socket

Se requiere saber si esta disponible el servicio de ejecución de operaciones y de transferencia de archivos **InterOPServer,** el código que valida esto, es el siguiente:

```
//Variable de comprobación de puerto abierto
boolean isOpenSocket = (System.getProperty("ORACLE_TRANSFER_MODE") != null ?
System.getProperty("ORACLE_TRANSFER_MODE").equals("SOCKET") : false)
        && (System.getProperty("ORACLE_TRANSFER_BUFFER_SIZE") != null)
        && (System.getProperty("EVR_INTER_OP_ORACLE_PORT") != null);
//Inicialización del objeto de conexión
socketLinker = new
SocketLinker(System.getProperty("EVR_INTER_OP_ORACLE_HOST"),
Integer.parseInt(System.getProperty("EVR_INTER_OP_ORACLE_PORT")),
communicationManager);
//Comprobación de puerto abierto
isOpenSocket = isOpenSocket ? socketLinker.isOpen() : false;
```

6.3.4.3 Transferencia de archivos

Hice las validaciones correspondientes para seleccionar en orden de preferencia el servicio por cual se enviarán los archivos, para cumplir con ello se describe el siguiente código, donde el orden de preferencia es SOCKET, FTP y SSH al final, pues SSH es el servicio más lento y SOCKET el más rápido por el hecho de contar con compresión de datos en la transferencia

```
//configurando la mejor conexión disponible
if (isOpenSocket) {
    transferChannel = socketLinker.connect() ? "SOCKET" : "FAIL";
} else if (System.getProperty("ORACLE_TRANSFER_MODE").equals("FTP")) {
    transferChannel = ftpLinker.connect() ? "FTP" : "FAIL";
```

```
} else if (System.getProperty("ORACLE_TRANSFER_MODE").equals("SSH")) {
    transferChannel = sshLinker.connect() ? "SSH" : "FAIL";
}
//enviando archivos por servicio configurado anteriormente
if (transferChannel.equals("SOCKET")) {
    isSend =
socketLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(configFile).get
AbsolutePath(), target + configFile);
    if (isSend) {
        isSend =
socketLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(fileName).getA
bsolutePath(), target + fileName);
    }
} else if (transferChannel.equals("FTP")) {
    ftpLinker = new FtpLinker(c.decode(Parameters.SYSUSR.getParam(),
Parameters.KEY.getParam()), c.decode(Parameters.SYSPASS.getParam(),
Parameters.KEY.getParam()), System.getProperty("EVR_INTER_OP_ORACLE_HOST"),
Integer.parseInt(System.getProperty("EVR_INTER_OP_ORACLE_PORT_FTP")),
communicationManager);
    isSend =
ftpLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(configFile).getAbs
olutePath(), target + configFile);
    if (isSend) {
        isSend =
ftpLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(fileName).getAbsol
utePath(), target + fileName);
    }
} else if (transferChannel.equals("SSH")) {
    isSend =
sshLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(configFile).getAbs
olutePath(), target + configFile);
    if (isSend) {
        isSend =
sshLinker.sendPersist(nestEnviromentManager.getEphemeralScriptFile(fileName).getAbs
olutePath(), target + fileName);
    }
}
```

6.3.4.4 Ejecución de operación SQLLOADER

Una vez asegurado el envío de archivos CTL y SQL, la siguiente operación es la ejecución del archivo .ctl encargado de la configuración de la carga de los datos contenidos en el archivo .sql, la forma sistemática queda descrita en el siguiente código:

```
//Valida que los archivos hayan sido enviado correctamente
if (!transferChannel.equals("FAIL") && isSend) {
```

```
    //Construcción de comando BASH
    //Posicionamiento en directorio de ejecución
    commandString.append("cd " + "\"").append(target).append("\"" + " && ");
    //Asignación de variables de entorno
    commandString.append("ORACLE_SID=" + parameters.getSid() + " && ");
    commandString.append("export ORACLE_SID && ");
    commandString.append("PATH=").append(Parameters.PATH.getParam()).append(" &&
");
    commandString.append("export PATH && ");

commandString.append("ORACLE_HOME=").append(Parameters.HOME.getParam()).ap
pend(" && ");
    commandString.append("export ORACLE_HOME && ");
    commandString.append("NLS_LANG=").append("\"mexican
spanish_mexico.WE8ISO8859P1\"").append(" && ");
    commandString.append("export NLS_LANG && ");

    //Reporta cantidad total de líneas de arhivo SQL
    commandString.append("wc -l ").append(fileName + " ").append("|tail -1 && echo ToTaL
&& ");

    //Reporta estado de la ejecución
    this.communicationManager.trace("sqlldr " + parameters.getUsr() + "/" +
parameters.getPsw() + " " + "\"" + configFile + "\"");
    commandString.append("sqlldr " + parameters.getUsr() + "/" + parameters.getPsw() +
"@" + parameters.getHost() + "/" + parameters.getSid() + " " + "\"" + configFile + "\"");

    this.communicationManager.openConext(currentSendAndLoad);
    this.communicationManager.updateCurrentMessage("[SQL-Loader] Status", "Espere,
se están realizando operaciones previas a la carga...");
    sshLinker.execSQLLoader(commandString.toString(), "\"" + target + "\"");

    //Obtención de operaciones rechazadas
    loaderLog = sshLinker.execResponse("wc -l " + target + fileName.substring(0,
fileName.lastIndexOf(".sql")) + ".bad", "Conteo de instrucciones rechazadas");
    if (!loaderLog.toString().isEmpty()) {
        this.communicationManager.updateCurrentMessage("Inserciones BAD", "Total de
instrucciones corruptas [" + loaderLog + "].",
CoreStatic.MSG_POINT_STATS.PROC_END_OK);
        this.communicationManager.closeContext();

        //Generación de reporte Log
        loaderLog = sshLinker.execResponse("cat " + target + fileName + ".log", "Contenido
de Log SQLloader");
        this.communicationManager.updateCurrentMessage("SQL-Loader Error-Log",
loaderLog.toString(), CoreStatic.MSG_POINT_STATS.I_FAIL);
        this.communicationManager.closeContext();
    }
} else {
    throw new Exception("Error de envío, los archivos no fueron transferidos.");
```

```
}
```

6.4 Análisis de Modelo Genérico de Base de Datos

La diversidad de datos y sus dimensiones crearon la necesidad de unificar el modelo en el que se guarda la información, el requerimiento surge al no poder soportar la cantidad de datos que las áreas no sistematizadas generan, estas utilizan recursos informáticos que van desde archivos Excel hasta Bases de Datos, pero la información no tiene un modelo estándar donde puedan concentrarse para su posterior procesamiento o intercambio con otros consumidores.

Para solventar esta problemática el Instituto creó un modelo de Base de Datos, con la suficiente flexibilidad para contener cualquier tipo de información bajo cualquier estructura y de cualquier tamaño.

El modelo que generó el Instituto es muy semejante a un Cubo OLAP, había que generar un análisis de su estructura para la generación de procesos de manipulación de los datos que podría contener, para ello generé este análisis de sus relaciones con el fin de abstraer su definición.

6.4.1 Tipo de Modelo

Es un modelo donde la tabla central es la que contiene a las cifras y está rodeada por el conjunto de características que puede poseer un dato del tipo cifra.

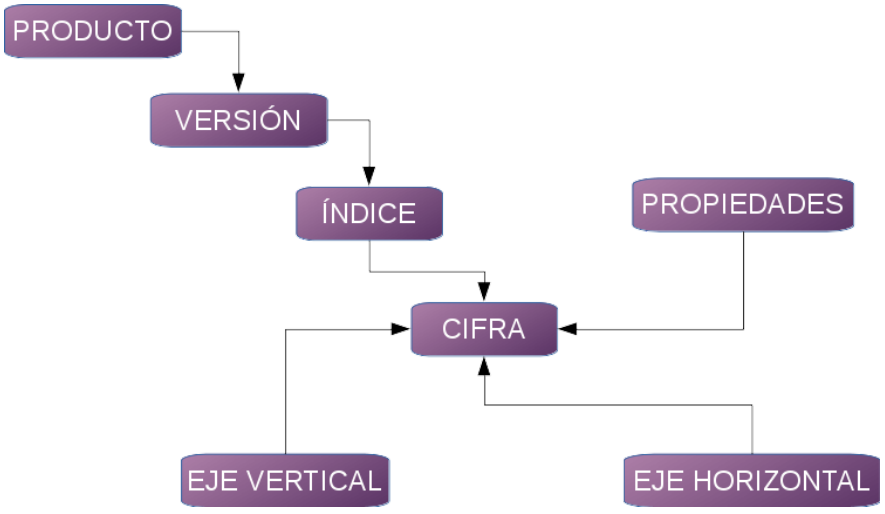

Figura 13. Estructura de Modelo Genérico

6.4.2 Estructura de Datos

La principal estructura que se encuentra en este modelo es de tipo árbol, se emula asignado relaciones hacia arriba entre registros, esto es que a cada elemento se le asigna un padre en caso de que lo tenga, o un elemento nulo o vacío para el caso de no tener padre o nodo superior.

Las estructuras en árbol son:

- Producto
- Versión
- Índice
- Árbol horizontal
- Árbol vertical

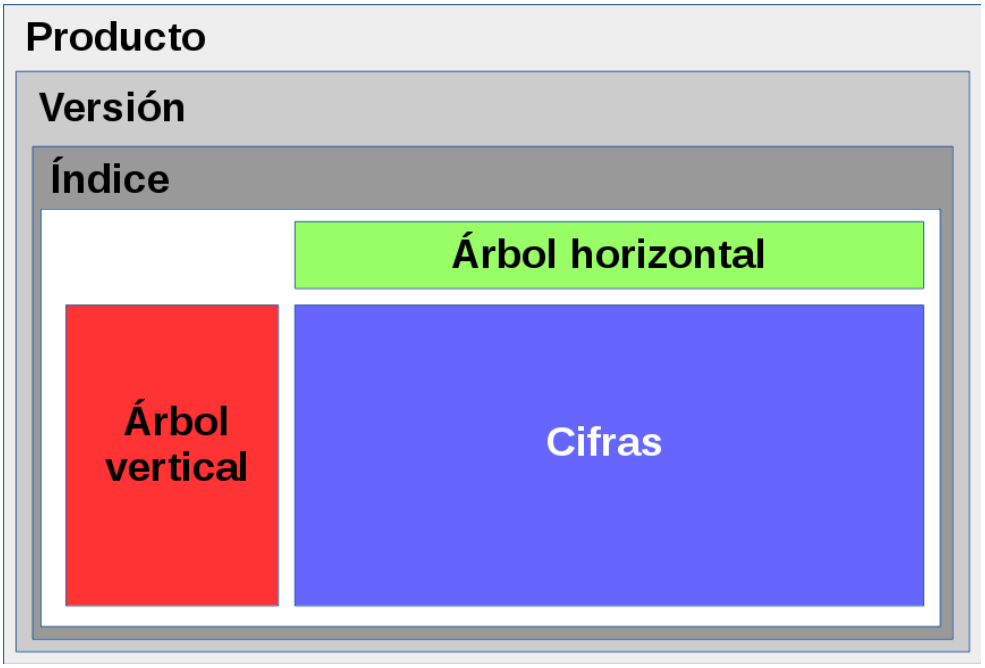

Figura 14. Agrupación de Modelo Genérico

6.5 Transformación de estructuras arbóreas de Modelo Genérico

Una vez comprendida la estructura del Modelo Genérico generé los procedimientos necesarios para extraer las relaciones, significados y valores. Fue necesario traducir el

modelo en estructuras de datos Java para su manipulación en WebServices, pues estos serían consumidos por recursos web.

6.5.1 Estructura arbórea de Modelo Genérico

En Java formé una estructura de datos que permite almacenar elementos y guardar la relación que existe entre ellos a manera de contenedores, donde un elemento forma parte de los objetos de otro elemento.

Las clases que creé para contener otros objetos de la misma clase, son hechas para construcción de columnas y de filas, también formé una estructura del tipo tabla que podrá contener filas y columnas para ser contenidas en una sola estructura de datos.

6.5.2.1 Tabla

Figura 15. Estructura Java para una tabla

Esta estructura se encarga de contener estructuras Columna y Fila al mismo tiempo, en su conjunto forman la estructura TABLA.

La clase en JAVA que permite interpretar el tipo de estructura, está formada principalmente por dos Mapas con llaves String y valores Column y Row, respectivamente, el siguiente código describe su formación:

```java
public class Table extends Primal implements Cloneable, Serializable {

    //Estructuras de almacen para filas y columnas
    LinkedHashMap<String, Column> columns;
    LinkedHashMap<String, Row> rows;

    //Inicialización de tabla
    public Table() {
        super();
        this.columns = new LinkedHashMap<String, Column>();
        this.rows = new LinkedHashMap<String, Row>();
    }

    //Agregado de columna
    public void addColumn(String key, Column column) {
        if (key != null && column != null) {
            this.columns.put(key, column);
        }
    }

    //Obtención de una columna según su posición
    public Column getColumnByPos(Integer pos) {
        for (Map.Entry<String, Column> column : columns.entrySet()) {
            if (pos.equals(column.getValue().getColumnPos())) {
                return column.getValue();
            }
        }
        return null;
    }

    //Agregado de fila
    public void addRow(String key, Row row) {
        if (key != null && row != null) {
            this.rows.put(key, row);
        }
    }

    //Obtención de fila según su posición
    public Row getRow(String keyPos) {
        return this.rows.get(keyPos);
    }
}
```

6.5.2.2 Fila

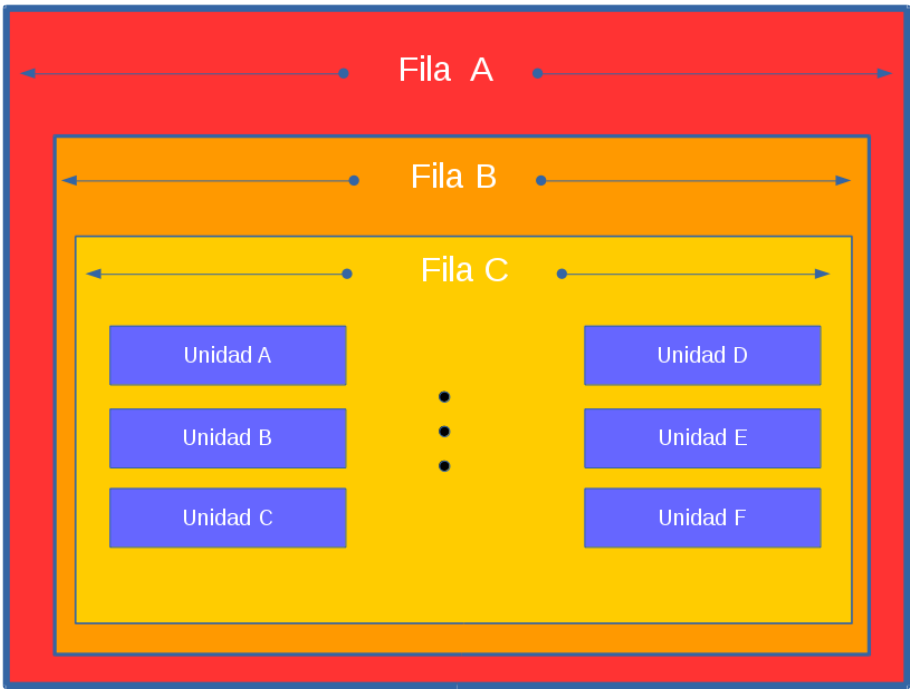

Figura 16. Estructura Java para una Fila

Esta estructura por sí sola representa a una Fila, pero tiene la capacidad de contener más elementos de su propio tipo (Fila), y cada una de estas últimas podrá contener más filas, esto permite la creación de descendencias prácticamente sin límite.

Cada elemento Fila, contiene a su vez Unidades, estas se describen al final, de este apartado.

Para el ejemplo mostrado, la ***Fila A*** almacena a la ***Fila B***, donde la ***Fila B*** almacena a la ***Fila C***.

```
public class Row extends Primal implements Cloneable, Serializable {
   //Contenedor de datos de Fila
   public LinkedHashMap<String, Unit> units;

   //Inicialización de Fila
   public Row() {
      super();
      this.units = new LinkedHashMap<String, Unit>();
   }

   //Agregado de dato para Fila
   public void addUnit(String key, Unit unit) {
      if (key != null && unit != null) {
         this.units.put(key, unit);
```

```
        }
    }

    //Obtención de dato contenido en Fila
    public Unit getUnit(String keyPos) {
        return this.units.get(keyPos);
    }

    //Obtención de todos los datos contenidos en una Fila
    public LinkedHashMap<String, Unit> getUnits() {
        return this.units;
    }
}
```

6.5.2.3 Columna

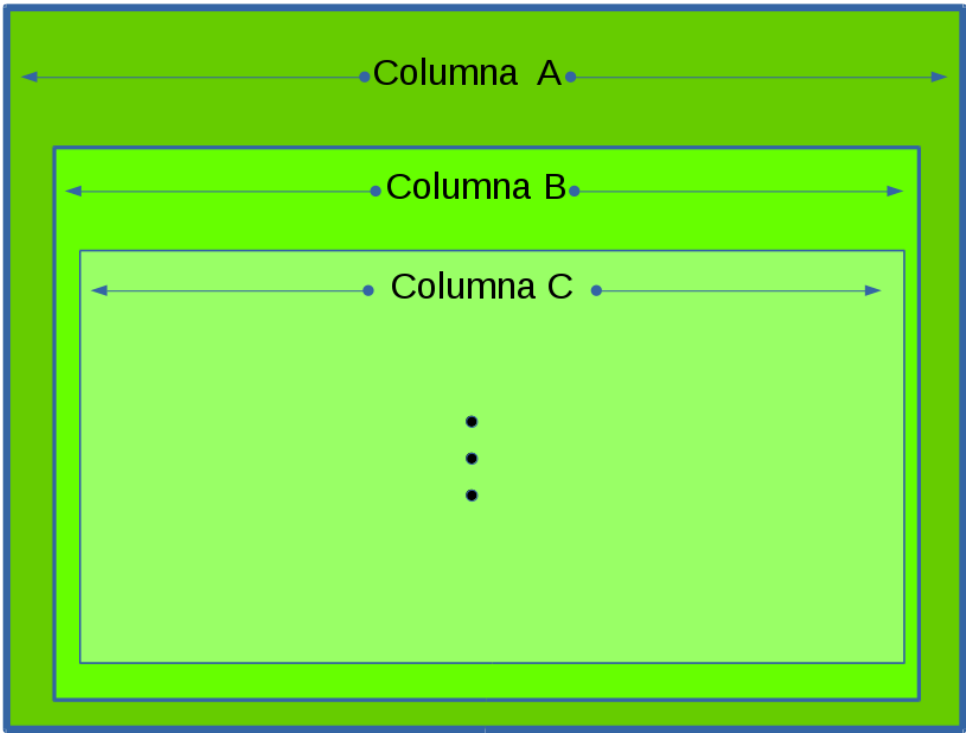

Figura 17. Estructura Java para una columna

Esta estructura por sí sola representa a una Columna, pero tiene la capacidad de contener más elementos de su propio tipo (Columna), y cada una de estas últimas podrá contener más columnas, esto permite la creación de descendencias prácticamente sin límite.

Para el ejemplo mostrado, la ***Columna A*** almacena a la ***Columna B***, donde la ***Columna B*** almacena a la ***Columna C***.

La estructura encargada del almacén de datos para una columna, es el siguiente:

```
public class Column extends Primal implements Cloneable, Serializable {
    //Elementos de identificación de columna, Tipo, Posición y Nombre
    private CoreStatic.DATABASE_COLUMN_TYPE dataBaseColumnType;
    private Integer columnPos;
    private String dbName;

    //Inicialización de Columna
    public Column() {
        super();
        this.columnPos = null;
        this.dataBaseColumnType = dataBaseColumnType.UNDEFINED;
        this.dbName = "";
    }

    //Obtención de tipo de columna
    public CoreStatic.DATABASE_COLUMN_TYPE getDataBaseColumnType() {
        return dataBaseColumnType;
    }

    //Asignación de tipo de columna
    public void setDataBaseColumnType(CoreStatic.DATABASE_COLUMN_TYPE
dataBaseColumnType) {
        this.dataBaseColumnType = dataBaseColumnType;
    }

    //Obtención de posición de columna
    public Integer getColumnPos() {
        return columnPos;
    }

    //Asignación de posición de columna
    public void setColumnPos(Integer columnPos) {
        this.columnPos = columnPos;
    }
}
```

6.5.2.4 Unidad

Las unidades son encargadas de almacenar el valor de un dato de cualquier naturaleza, para ello se contemplan datos del tipo:

1. Double
2. String
3. Object

De esta forma consideré los dos tipos más comunes, que son el numérico y el de conjunto de caracteres, para los demás tipos de datos, todos deben poder estar contenidos en tipo Object, que finalmente es la unidad fundamental en JAVA.

La interfaz en la que se basan los tipos finales, es la siguiente, está encargada de definir el comportamiento de cualquier implementación de tipos de datos:

```
public interface Unit {
    public void setData(Double data);
    public void setData(String data);
    public void setData(Object data);
    public void setNullData();
    public String getDataAsString();
    public double getDataAsDouble();
    public long getDataAsLong();
    public boolean isNull();
    public boolean isNumeric();
    public Object getDataAsObject();
    public Class getTypeOf();
    public Unit replica();
}
```

6.5.3 Estructura arbórea JSON

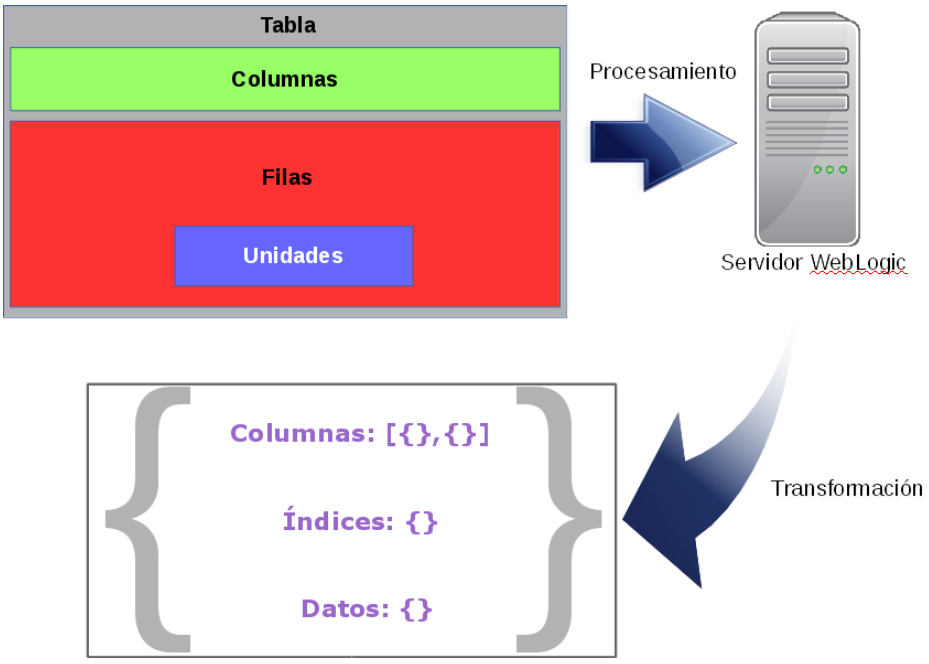

Figura 18. Transformación de Modelo Genérico a estructuras Java

Establecí una estructura JSON que se construye a partir de los elementos contenidos en una tabla, la cadena resultante se ajustará al objeto esperado por treepanel de ExtJs.

Para una estructura JSON, este requerimiento es natural, pues la formación de este tipo de datos es implementada de forma directa, pues un objeto puede estar contenido dentro de otro, cada uno con un conjunto de propiedades independientes, la forma de organizar una Tabla para ser representada por el navegador, es la siguiente:

```
{
   //Arreglo de columnas
   columns:[
      {
         //Definición de tipo de columna en modo árbol
         xtype: 'treeColumn',
         text: 'Concepto',
         dataIndex: 'text'
      },
      {
         //Definición simple de columna
         text: 'Producción',

         //Definición de descendencia de columna 'Producción'
         columns: [

            //Definición de arreglo de columnas de descendencia
            {
               text: '2003',
               dataIndex: 'x1'
            },
            {
               text: '2004',
               dataIndex: 'x2'
            },
            {
               text: '2005',
               dataIndex: 'x3'
            }
            .
            .
            .
         ]
      }
   ],

   //Conjunto de índices de columnas
   store: {'text','x1','x2','x3',...},

   //Objeto de descripción de datos
   data: {
```

```
    //Definición de arreglo de índices de columnas
    fields: ['text','x1','x2','x3',...],

    //Elemento raíz del árbol
    root: {

      //Inicio de descendencia de nodo raíz
      children: [
        {

          //Datos asignados a cada índice de columna
          text: 'Total',x1: '12,544,482', x2: '14,270,034', x3: '15,635,318', ...

          //Arreglo correspondiente a la descendencia el primer nodo
          children: [
            {
              text: 'Sector público', x1: '2,045,662', x2: '2,405,030', x3: '2,777,239', ...

              //Arreglo correspondiente a la descendencia el segundo nodo
              children: [
                {
                  text: 'Gobierno General', x1: '873,434', x2: '930,476', x3:
'1,012,096', ...

                  //Arreglo correspondiente a la descendencia el tercer nodo
                  children: [
                    {
                      text: 'Gobierno central', x1: '262,374', x2: '258,082', x3:
'269,648', ...

                      //Arreglo correspondiente a la descendencia el cuarto nodo
                      children: [
                        {
                          text: 'Gobierno Federal Integrado', x1: '204,476', x2:
'198,178', x3: '206,472', ...
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        }
      ]
    }
  }
}
```

6.6 Visor Web de Modelo Genérico

Para la visualización de los datos que ahora podía arrojar un WebService, desarrollé una plataforma para mostrar la información, una vez más, respetando su estructura y significado, además de sus valores tal cual fueron definidos desde su alimentación.

Este visor de datos, debía tener la capacidad de mostrar grandes volúmenes de información, además de la capacidad de filtrar, estructurar, agrupar y presentar claramente los datos.

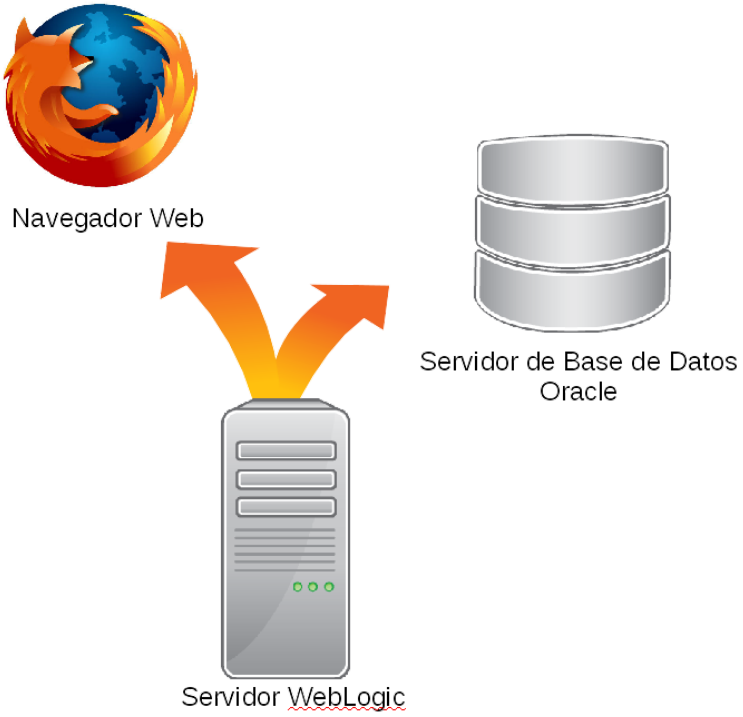

Figura 19. Intervención central de Servidor de Aplicaciones

6.6.1 Agrupamientos

- Producto: elemento de mayor jerarquía que permite agrupar la información por un proyecto o producto independiente.
- Versión: versión de producto para agrupación de información fuente y resultados por ejemplo.
- Índice: árbol de navegación que permite direccionar a los datos procedentes de un archivo Excel o reporte de resultados.
- Eje vertical: referencia Y o vertical para ubicar una cifra.
- Eje horizontal: referencia X u horizontal para ubicar una cifra.

La vista que desarrollé para la presentación de productos y versiones es la siguiente:

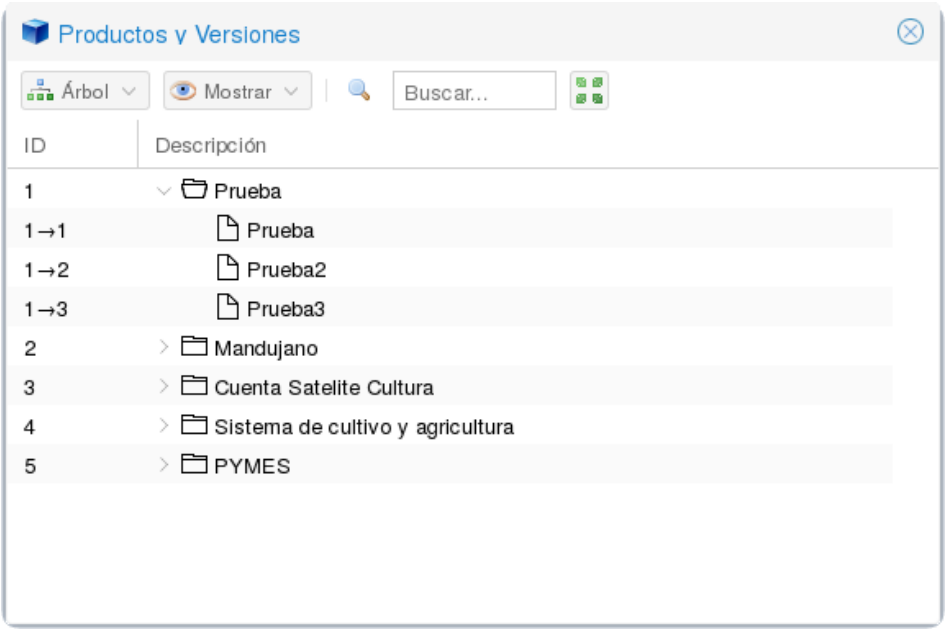

Figura 20. Vista de relación Producto/Versión

La vista que desarrollé para la presentación de Índice es la siguiente:

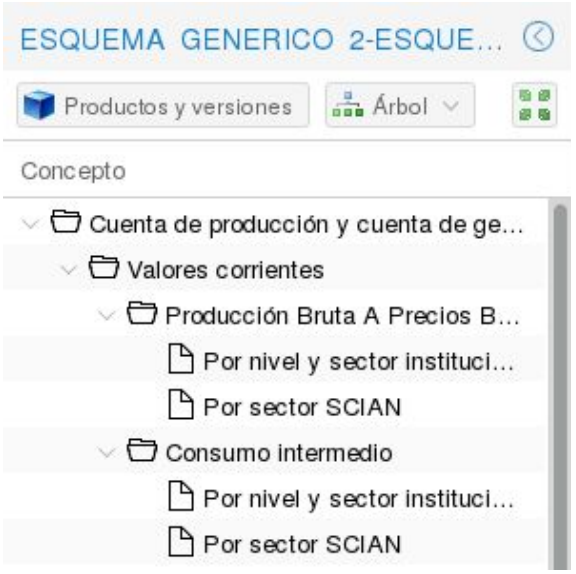

Figura 21. Vista de árbol Índice

La vista que desarrollé para la presentación de los ejes, Vertical y Horizontal, es la siguiente:

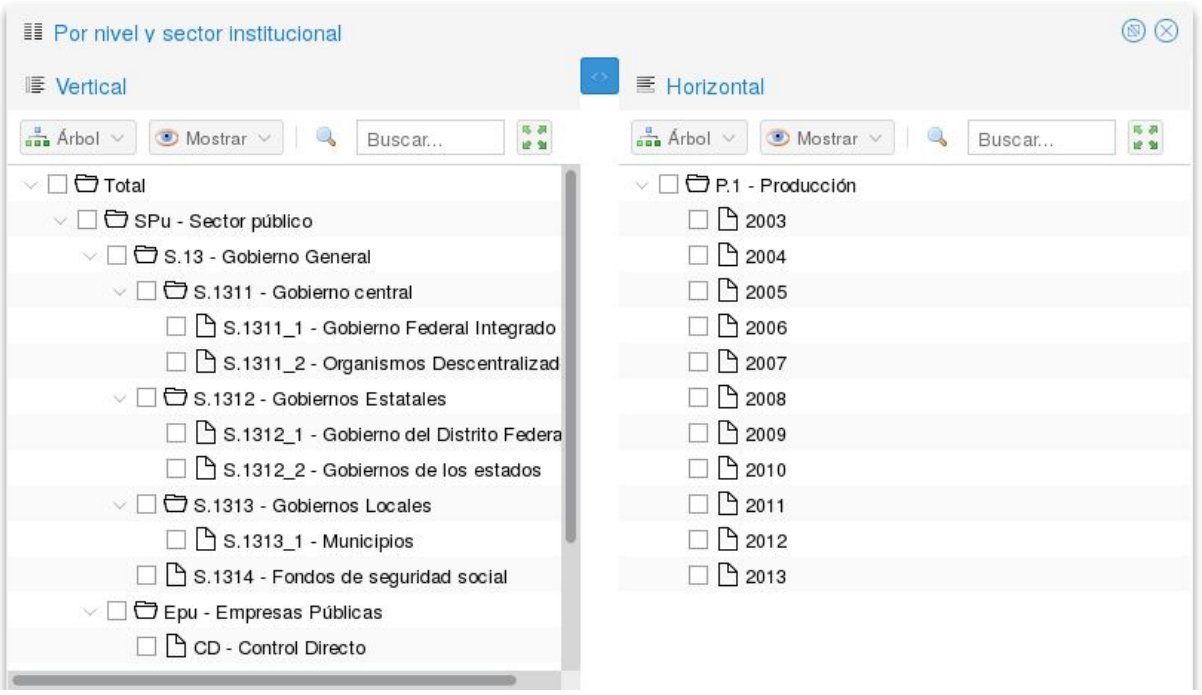

Figura 22. Vista para filtrado de Cruce de Datos

6.6.2 Estructuración

Transformé las estructuras arbóreas en base de datos en estructuras JSON, que son directamente enviadas al navegador para ser interpretadas por Ext JS Versión 6.5.1.

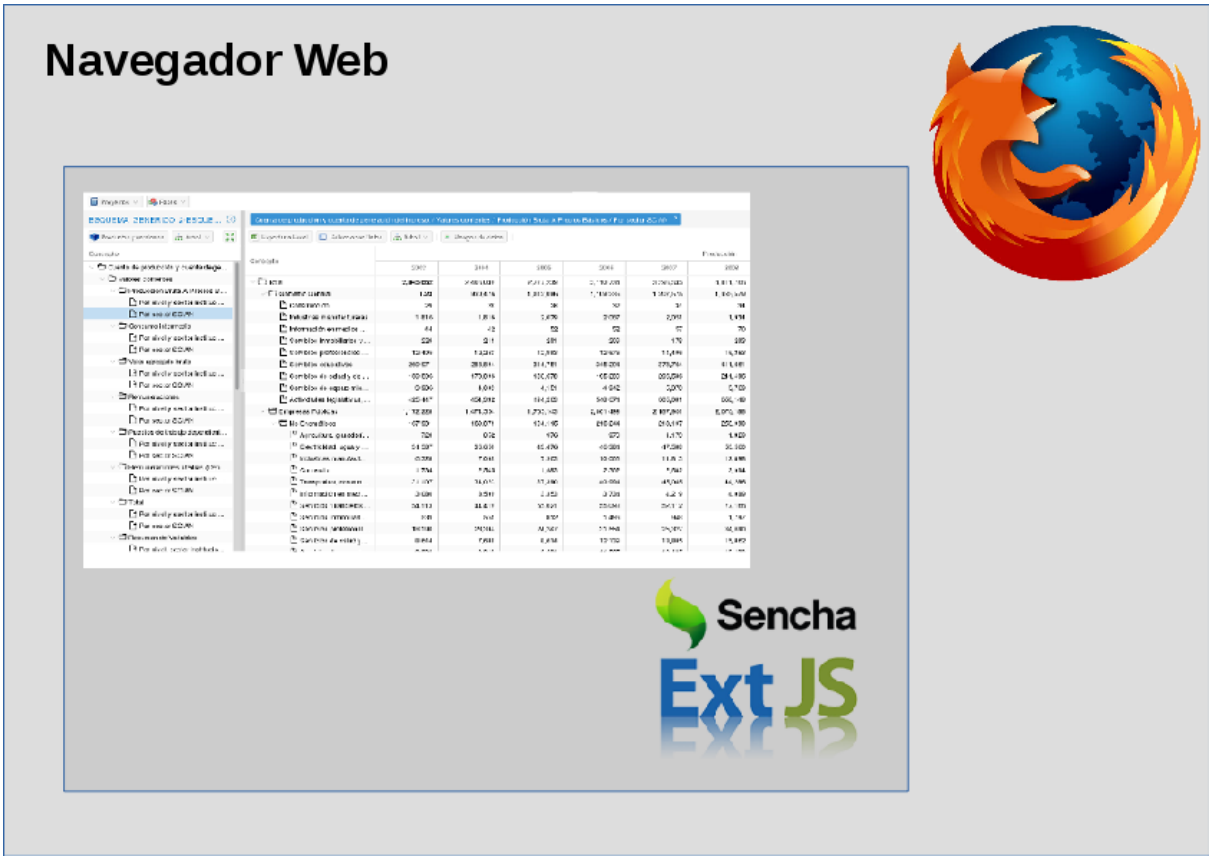

Figura 23. Agrupación de aplicaciones para cliente final

6.6.3 Conversión de estructuras

Una vez realizada la agrupación y estructuración de los datos convertí el objeto Java en objeto JSON para poder devolver como respuesta de parte del WebService.

El procedimiento general para la transformación de datos, consiste en tres puntos

- Llamada al Servicio Web, esta nombrada como ***TreeCrossService*** en el desarrollo d este trabajo, el método nombrado **getResponse()** es el encargado de atender las solicitudes provenientes de los clientes Web, en este punto validé las peticiones y la parametría, éstas son sometidas a procedimientos de seguridad para mitigar los riesgos conocidos, como los son:
    - Inyección de SQL, es un tipo de ataque destinado realizar consultas, inserciones o modificaciones de cualquier tipo a una Base de Datos a través de un sitio Web.
    - Sobrecarga de parámetros, es un ataque cuyo fin es incrementar la longitud de la parametría que recibe un servidor para provocar su desestabilización.
- Construcción de Tabla, está nombrada como ***JsonConstructs*** en el desarrollo de este trabajo, el método nombrado **getAdvancedCrossTableTreeMeta()** es el encargado de leer el contenido que solicitó el cliente, los datos son leídos directamente de la Base de Datos, estos son procesados con el fin de llenar la estructuras que fueron descritas anteriormente:
    - Unidad
    - Fila
    - Columna
    - Tabla
- Transformación de Tabla en JSON, finalmente esta clase que en el desarrollo presentado por este trabajo, está nombrada como **AdvancedTreeTransferExt**, es la encargada de transformar la estructura Tabla en el formato que requiere TreePanel de Ext JS a través del método nombrado como **getJsonTreeFromData()**.

El siguiente código representa los tres puntos principales anteriormente descritos, en ellos se encuentran las tres llamadas para lograr la transformación de los datos y poder ser entregado al cliente:

```
//Nombramiento del servicio Web
@Path("treeCrossService")
public class TreeCrossService {
   //Tipo de respuesta REST
   @POST
   @Produces({"application/json"})
   //Primer método, encargado de atender las solicitudes hechas por los clientes Web
   public Response getResponse(String in) throws Exception {
      ResponseBuilder response = Response.ok();
      try {
```

```
        //Segundo método, encargado del llenado de estructuras JAVA
        final AdvancedTable table =
jsonConstructs.getAdvancedCrossTableTreeMeta(queryXY, queryX, queryY, conn,
oracleDataBaseManager, prevTime);

        //Formación de stream para responder al cliente sin almacenar demasiados datos
en memoria de JAVA
        StreamingOutput stream = new StreamingOutput() {
            @Override
            public void write(OutputStream output) throws IOException,
WebApplicationException {
                BufferedWriter br = new BufferedWriter(new OutputStreamWriter(output,
"UTF-8"));
                try {
                    //Tercer método, encargado de la transformación de estructuras JAVA en
JSON
                    new AdvancedTreeTransferExt(false, true,
true).getJsonTreeFromData(table, br);
                    br.flush();
                } catch (Exception ex) {
                }
            }
        };
        response = Response.ok(stream);
      } catch (Exception e) {
      }

      return response.build();
   }
}
```

6.6.4 Procedimiento de formación de estructuras

A continuación se describe el procedimiento principal para la formación de estructuras, está nombrado como **getAdvancedCrossTableTreeMeta()**.

Este método es utilizado por todos los componentes de salida de datos, como lo son la exportación de CSV, BufferedStore y formato JSON para Ext JS.

Se ha diseñado especialmente para poder soportar cada uno de los tipos de visualización, el principal motivo de sobrecargar este método en cuanto a sus usos, es por mantenimiento y por el uso recurrente de la misma estructura de datos, asi cuando solo se desea cambiar el tipo de salida ya sea para un tipo de archivo o una salida Web, utilicé la misma construcción cambiando únicamente el formato de salida de los datos.

Para lograr unificar el uso de este método por los diferentes formatos de salida generé una lógica que describe la secuencia del uso de hilos de ejecución dependiendo de la solicitud

que se realice, es aquí donde el procedimiento se identifica como proceso en paralelo, en algunos casos es posible ejecutar dos hilos simultáneamente, en otros es necesario que la construcción de un hilo esté finalizada para continuar con la construcción del siguiente.

Bajo este método se realizan operaciones como el barrido de la base de datos en las tablas principales Eje Horizontal, Eje Vertical y Cifra, también se crean las estructuras de datos basadas en HashMap y ArrayList de JAVA, con la finalidad de relacionar los datos que son leídos.

El siguiente diagrama representa el diseño de ejecución de este procedimiento, donde a partir del proceso principal se generan tres Hilos de ejecución que realizan tareas simultáneas con el fin de acelerar la respuesta del servidor, esto en lugar de tener un proceso secuencial donde cada tarea deba ser seguida o precedida de otra.

Figura 24. Procesamiento paralelo de estructuras

Al realizar este diseño incrementé el uso de procesamiento pero aceleré el tiempo de respuesta aprovechando en gran medida el uso de la base de datos, la red de datos y finalmente las capacidades del servidor de aplicaciones.

El siguiente diagrama representa la secuencia en la que son ejecutados los hilos de acuerdo al tipo de salida que se solicite.

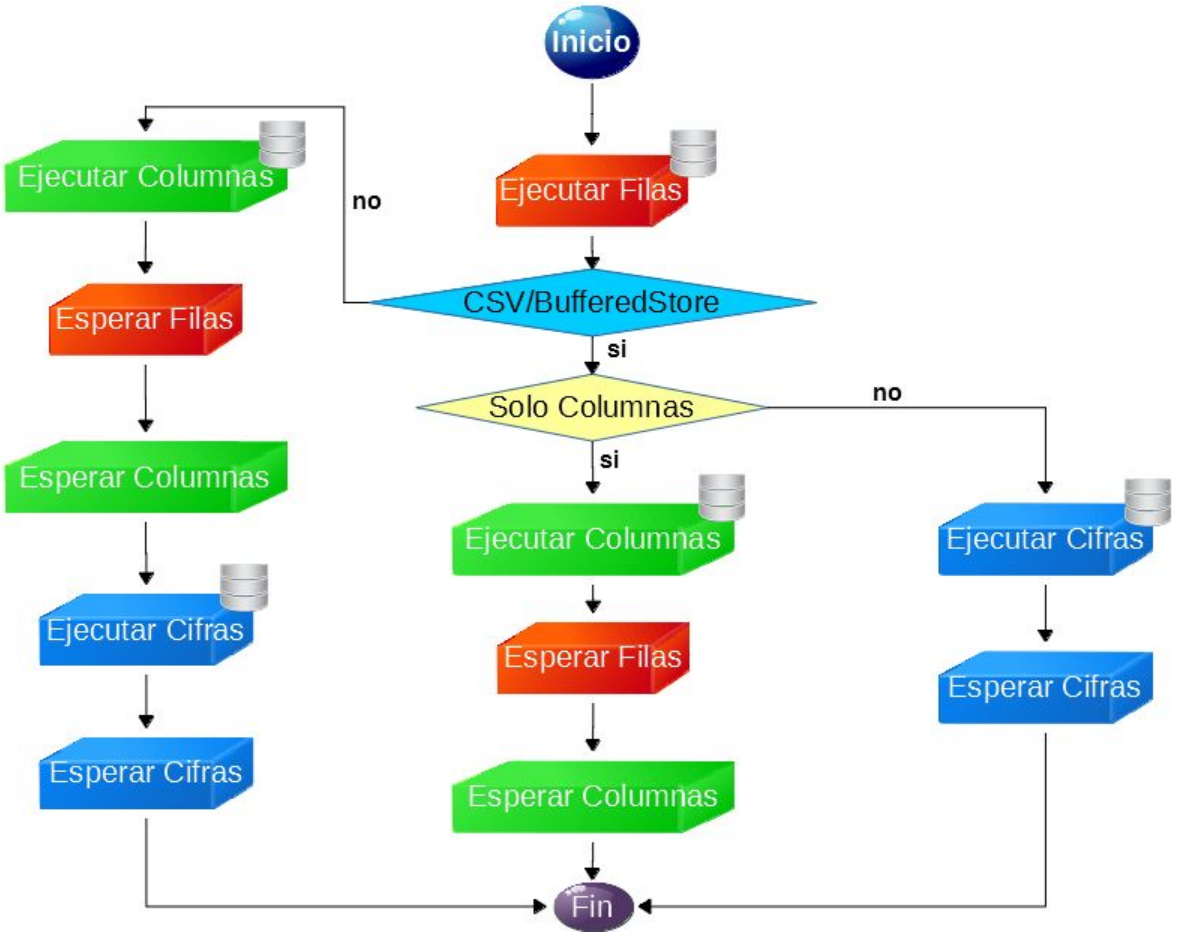

Figura 25. Ejecución paralela para formación de estructuras Java

En el siguiente código se implementa la lógica del diagrama anterior:

```
//Inicio de hilo constructor de filas
rowsThread.start();
if (!tableMode || onlyColumns) {
    //Inicio de hilo constructor de columnas
    columnsThread.start();
}

try {
    synchronized (rowsThread) {
        //Espera de hilo de filas
        rowsThread.wait();
    }

    //Se agregó esta condición porque cuando es editable se requiere que los dos ejes ya
estén armados
    if (editable && !tableMode) {
        synchronized (columnsThread) {
            //Espera de hilo de columnas
            columnsThread.wait();
        }}
```

```
    if (!tableMode || !onlyColumns) {
        //Inicio de hilo constructor de cifras
        cipherThread.start();
    }
} catch (Exception e) {
    miscUtils.printSimpleError(e, this.getClass().getName(), MiscUtils.WARNING);
}

if (!tableMode || !onlyColumns) {
    try {
        synchronized (cipherThread) {
            //Espera de hilo de cifras
            cipherThread.wait();
        }
    } catch (Exception e) {
        miscUtils.printSimpleError(e, this.getClass().getName(), MiscUtils.WARNING);
    }
}

if (!tableMode || onlyColumns) {
    try {
        synchronized (columnsThread) {
            //Verificación de término de columnas
            columnsThread.join();
        }
    } catch (Exception e) {
        miscUtils.printSimpleError(e, this.getClass().getName(), MiscUtils.WARNING);
    }
}
```

Para la implementación de Hilos Java utilicé la clase principal de Thread, a continuación se describe la codificación, donde son declarados para en algún momento ser llamados a ejecución por parte del algoritmo anteriormente descrito:

```
//Hilo de formación de filas
Thread rowsThread = new Thread(new Runnable() {
    @Override
    public void run() {
        ...
    }
});

//Hilo de formación de columnas
Thread columnsThread = new Thread(new Runnable() {
    @Override
    public void run() {
        ...
    }
});
```

```
//Hilo de formación de cifras
Thread cipherThread = new Thread(new Runnable() {
    @Override
    public void run() {
        ...
    }
});
```

6.6.5 Formato de cifras

Debido a que las salidas o vistas que serán generadas son esencialmente para usuarios finales es requerido ajustar el formato de las cifras a un estándar fijado por la empresa, donde los valores numéricos deben estar separados por miles de unidades.

Para este propósito ocupe la clase java.text.NumberFormat, recibe como parámetro un número y genera una cadena de caracteres con el formato separado por comas, también se puede especificar la cantidad de elementos decimales que deberá contener la cadena resultante.

El código siguiente describe su utilización:

```
//Declaración de variable
NumberFormat numberFormat = NumberFormat.getNumberInstance();
//Indicación de cantidad de decimales a tomar en cuenta
numberFormat.setMinimumFractionDigits(5);
//Número
double number = 193.657979979;
//Salida de texto con formato numérico
System.out.println(numberFormat.format(number));
```

6.6.6 Exportación Excel (XLSX)

Una funcionalidad de gran importancia para la visualización de datos, es la posibilidad de exportar los datos a un medio más amigable como es el formato Excel (XLSX).

Para lograr la creación de este formato, utilicé la biblioteca Apache POI (org.apache.poi), en su implementación de transmisión o streaming con la finalidad de ahorrar la máxima memoria JAVA posible.
El procedimiento general para la transformación de datos, consiste en dos puntos:

- Llamada al Servicio Web, está nombrada como **ExcelExportService** en el desarrollo de este trabajo, el método nombrado **getResponse()** es el encargado de atender las solicitudes provenientes de los clientes Web.
- Transformación de Tabla en Excel, esta es la clase responsable de generar un archivo XLSX, está nombrado como **ExcelConstructs** y el método responsable de generar el formato de salida es **formatSheetTree()**.

El procedimiento **formatSheetTree()**, también tiene un diseño de procesamiento en paralelo, pero solo para obtener los registros de la base de datos y depositarlos en estructuras que guardan la lógica en la que están relacionados los datos, principalmente para describir la estructura arbórea que guardan los ejes de navegación. Por esta razón se ejecutan los dos Hilos de eje Horizontal y Vertical al mismo tiempo, para que construyan los ejes antes de comenzar a asignar las cifras a los cruces de datos.

El siguiente diagrama representa la lógica de ejecución para la formación del archivo excel.

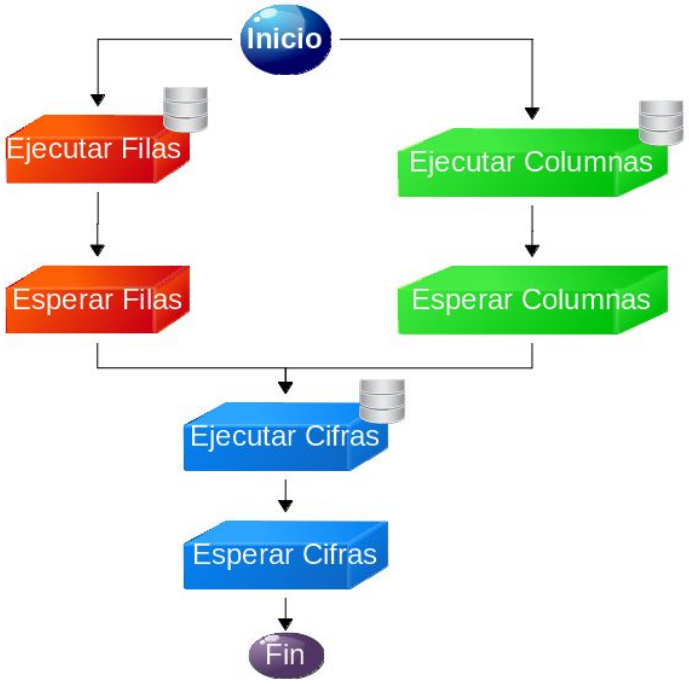

Figura 26. Ejecución paralela de formación de archivo Excel

El modelo de código encargado de esta tarea, es el siguiente:

```
//Hilo encargado de recuperar los datos de Base para eje Vertical
Thread threadTableTreeY = new Thread(new Runnable() {
   @Override
   public void run() {
      try {
         //Método de lectura, estructuración y almacenamiento de registros
         getTableTree(isTranspose ? queryX : queryY, conn, isTranspose ? "X" : "Y",
oracleDataBaseManager, tableTreeY, specialNodes);
```

```
        } catch (Throwable ex) {
            ex.printStackTrace();
        }
    }
});

//Hilo encargado de recuperar los datos de Base para eje Horizontal
Thread threadTableTreeX = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            //Método de lectura, estructuración y almacenamiento de registros
            getTableTree(isTranspose ? queryY : queryX, conn, isTranspose ? "Y" : "X",
oracleDataBaseManager, tableTreeX, specialNodes);
        } catch (Throwable ex) {
            ex.printStackTrace();
        }
    }
});

//Ejecución paralela de Hilos
threadTableTreeY.start();
threadTableTreeX.start();

//Código que realiza la espera del término de los hilos iniciados
try {
    if (threadTableTreeY.isAlive()) {
        synchronized (threadTableTreeY) {
            //Instrucción que indica al proceso actual que debe esperar hasta que
threadTableTreeY termine
            threadTableTreeY.wait();
        }
    }
    if (threadTableTreeX.isAlive()) {
        synchronized (threadTableTreeX) {
            //Instrucción que indica al proceso actual que debe esperar hasta que
threadTableTreeX termine
            threadTableTreeX.wait();
        }
    }
} catch (Exception e) {
    //Registro de errores
    miscUtils.printSimpleError(e, this.getClass().getName(), MiscUtils.ERROR);
};
```

6.6.6.1 Generación de formato XLSX

Una vez terminado el proceso anterior, es necesario transformar las estructuras de datos generadas al formato XLSX con ayuda de la biblioteca Apache POI.

El diseño de este procedimiento es:

1. Creación de libro de trabajo.

```
//Se utiliza la clase org.apache.poi.xssf.streaming.SXSSFWorkbook
SXSSFWorkbook wb = new SXSSFWorkbook();
```

2. Definición de fuentes y estilos para celdas.

```
//Crea un formato que es posible asignar a celdas para cambiar
DataFormat format = wb.createDataFormat();

//Crea una fuente color blanca y negrita
Font fontWhiteBold= wb.createFont();
//Agrega propiedad de Negrita
fontWhiteBold.setBoldweight(Font.BOLDWEIGHT_BOLD);
//Agrega propiedad de color Blanco
fontWhiteBold.setColor(IndexedColors.WHITE.getIndex());

//Crea un estilo de celda que después puede ser asignado a cualquier celda creada para
que tenga el estilo definido

//Creación de estilo
XSSFCellStyle yHeaderStyle = (XSSFCellStyle) wb.createCellStyle();
//Asigna el fondo de la celda
yHeaderStyle.setFillForegroundColor(new XSSFColor(new Color(165, 165, 165)));
//Asigna la alineación a la que será ajustado el texto de la celda
yHeaderStyle.setAlignment(CellStyle.ALIGN_CENTER);
//Indica el tipo de patrón que será dibujado como fondo, en este caso indica que el fondo
es de un color homogéneo o sólido, sin texturas.
yHeaderStyle.setFillPattern(CellStyle.SOLID_FOREGROUND);
//Asigna el estilo de la fuente que será reflejada por la celda
yHeaderStyle.setFont(font);

//Asignación de colores
yHeaderStyle.setBorderBottom(CellStyle.BORDER_THIN);
//Asignación de color de bordes de la celda
yHeaderStyle.setBottomBorderColor(IndexedColors.WHITE.getIndex());
yHeaderStyle.setBorderLeft(CellStyle.BORDER_THIN);
yHeaderStyle.setLeftBorderColor(IndexedColors.WHITE.getIndex());
yHeaderStyle.setBorderRight(CellStyle.BORDER_THIN);
yHeaderStyle.setRightBorderColor(IndexedColors.WHITE.getIndex());
yHeaderStyle.setBorderTop(CellStyle.BORDER_THIN);
```

```
yHeaderStyle.setTopBorderColor(IndexedColors.WHITE.getIndex());
```

3. Creación de hoja.

```
//El libro creado tiene la capacidad de generar hojas de trabajo
Sheet sheet = wb.createSheet();
```

a. Creación de comentarios.

```
//Creación de objetos previos al comentario
CreationHelper factory = wb.getCreationHelper();
Drawing drawing = sheet.createDrawingPatriarch();
ClientAnchor anchor = factory.createClientAnchor();
//Creación de comentario
Comment comment = drawing.createCellComment(anchor);
//Asignación de valores Fila/Columna donde será aplicado el comentario
comment.setColumn(cell.getColumnIndex());
comment.setRow(cell.getRowIndex());
//Creación de objeto contenedor de Texto con el formato aplicado
RichTextString str =
factory.createRichTextString(Double.toString(cell.getNumericCellValue()));
//Asignación del contenido de un comentario
comment.setString(str);
```

b. Creación de Filas.

```
//Creación de Filas, recibe como parámetro el número de fila comenzando desde 0 como
la primer fila de un hoja de cálculo
row = sheet.createRow(0);
row = sheet.createRow(1);
row = sheet.createRow(2);
...
```

i. Creación de Columnas o Celdas.

```
//Creación de columnas, recibe como parámetro el número de columna comenzando
desde 0 como la primer columna de una hoja de cálculo
cell = row.createCell(0);
cell = row.createCell(1);
cell = row.createCell(2);
...
```

c. Creación de regiones combinadas.

```
//Realiza la combinación de dos o más celdas, recibe como parámetro principal, un objeto CellRangeAddress, que a su vez recibe las coordenadas de un cuadrante, es decir el la primer y última fila, además de la primer y última columna, esto hace que se mezclen las celdas que se encuentran bajo este cuadrante.
sheet.addMergedRegion(new CellRangeAddress(firstRow, lastRow, firstColumn, lastColumn));
```

Una vez completado este proceso de transformación de las estructuras de datos, se comienza la transferencia del flujo de datos al nodo cliente, para esto utilicé la función **write()** de **SXSSFWorkbook**, el proceso de enviar el archivo resultante XLSX mediante flujos de bytes, se describe a continuación:

```
//Declaración de flujo de salida, este flujo transportará los datos del formato generado por POI
ByteArrayOutputStream outputStream = null;
try {
    //Declaración de Libro de trabajo
    SXSSFWorkbook wb = new SXSSFWorkbook(-1);
    //Transformación de información a estructuras de datos y generación de XLSX
    excelConstructs.formatSheetTree(wb, queryXY, queryX, queryY, conn, oracleDataBaseManager, prevTime);
    //Inicialización de flujo de bytes.
    outputStream = new ByteArrayOutputStream();
    //En este punto se solicita al libro de trabajo, dar salida a los datos por el flujo de bytes que se pasa como parámetro (outputStream)
    wb.write(outputStream);
    //Se libera el uso del recurso Libro de Trabajo
    wb.dispose();
    //Se libera la concentración de información en el flujo de datos
    outputStream.flush();
    //Se asigna el tipo de respuesta como un flujo de datos
    response = Response.ok(outputStream.toByteArray());
} catch (Exception e) {
    //En caso de fallo se responde con una respuesta vacía que puede ser interpretada por el cliente Web
    response = Response.ok();
    miscUtils.printSimpleError(e, this.getClass().getName(), MiscUtils.ERROR);
} finally {
    //En cualquier caso de éxito o fallo se cierran las conexiones y flujos con el fin de dejar libres los recursos de red y de servidor de aplicaciones
    oracleDataBaseManager.closeAllConnection();
    if (outputStream != null) {
        outputStream.flush();
        outputStream.close();
    }
}
```

```
//Finalmente se envía una respuesta al cliente con la información solicitada y un nombre de archivo
response.header("Content-Disposition", "attachment; filename=\"" + (conn.has("fileName") ? conn.getString("fileName").trim() : "SitioInterno.xlsx") + "\"");
return response.build();
```

La principal desventaja de encargar este procedimiento al Servicio Web, es que ocupa gran cantidad de memoria RAM, su capacidad bajo los parámetros establecidos por el servidor WebLogic 12c con el que se cuenta, se encuentra bajo el límite de 4 GB de memoria para la instancia que ejecuta el Visor Interno, donde cada proceso ó llamada de servicio puede alcanzar un máximo de 1.2 GB.

Con esta cantidad de memoria máxima por cada proceso logre optimizar el procedimiento para que pueda generar datos de hasta 1 millón de cifras.

6.6.7 Exportación CSV

La segunda modalidad de exportación de datos es CSV, un formato de texto plano, diseñado para guardar valores en forma de tabla, formado por un número de filas y un número de columnas.

La necesidad de crear este formato para la salida de datos es principalmente por la gran cantidad de datos que cada vez se requieren para ser exportados, los requerimientos han crecido al punto de generar archivos que contienen más de dos millones de cruces de información.

Se ha diseñado un procedimiento basado en procesamiento progresivo de los datos, que es atendido por el método antes descrito **getAdvancedCrossTableTreeMeta()** para generar las estructuras necesarias, es posible construir una porción de datos a la vez para formar el documento final con cada porción añadida.

La forma en que logre generar grandes cantidades de información a partir de páginas que son cada una un subconjunto de la totalidad de información que se solicita por el cliente, su generación está descrita en el siguiente diagrama:

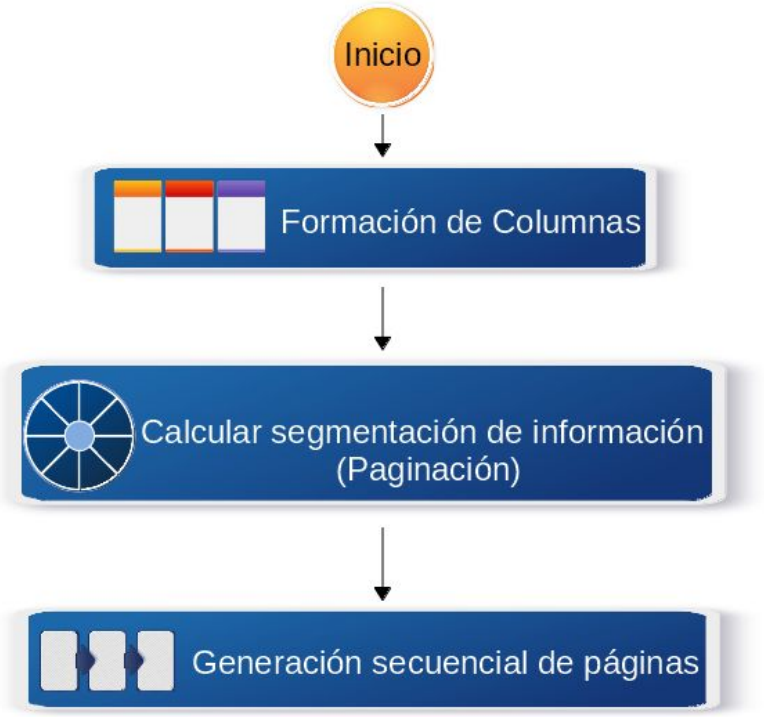

Figura 27. Proceso de formación de CSV

6.6.7.1 Formación de columnas

En esta etapa se realiza la consulta de información a la base de datos, generé estructuras arbóreas para relacionar la información, además obtuve dos datos indispensables para el cálculo de paginación, uno es el ancho o número de columnas y el otro es el largo ó número de filas.

```
//Generación de flujo final que será la respuesta al usuario
StreamingOutput stream = new StreamingOutput() {
    @Override
    public void write(OutputStream output) throws IOException, WebApplicationException {
        //Objeto encargado de escribir el flujo de datos
        BufferedWriter br = new BufferedWriter(new OutputStreamWriter(output, "CP1252"));
        //Objeto que genera el formato CSV
        CSVWriter csvWriter = new CSVWriter(br);
        try {
            //Parámetro que indica que solo se debe formar la estructura de columnas
            conn.put("onlyColumns", true);
            //Objeto que guarda las estructuras Fila/Columna resultantes de
getAdvancedCrossTableTreeMeta()
            AdvancedTable table =
jsonConstructs.getAdvancedCrossTableTreeMeta(jsonConstructs.getQueryCross(conn),
jsonConstructs.getQueryAxisX(conn), jsonConstructs.getQueryAxisY(conn), conn,
oracleDataBaseManager, new Date().getTime());
```

```
        //Extracción de estructura de datos para columnas
        HashMap tableColumns = table.getTableColumns();
        HashMap plainColumnTableByKey = table.getPlainColumnTableByKey();
        //Extracción de total de filas que contiene el cruce de datos
        conn.put("preCount",
Integer.parseInt(table.getRowByKey("total").getDataByKey("total").getAsString()));
...
```

Las estructuras de datos que guardan las columnas, se necesitarán más adelante para el posicionamiento de cada cifra en el lugar que le corresponde, hablando de una fila y una columna para cada cifra. Es por esto que las columnas son asignadas a una variable (tableColumns), no es necesario volver a construir los datos que no van a cambiar en la misma transacción.

6.6.7.2 Calcular segmentación de información (Paginación)

La paginación es el resultado de dividir la información para que pueda ser tratada cualquier cantidad de información sin afectar el rendimiento del servidor de aplicaciones así como los recursos de la red de datos.

El siguiente código muestra la obtención de este cálculo:

```
//Variable que define la cantidad mínima de 25 filas por página.
int defaultPageSize = 25;
int pageSize = defaultPageSize;
//Cantidad máxima de cifras de que puede contener una página
//Este es un valor predefinido de 100,000, se obtuvo de acuerdo a las capacidades
//del servidor de aplicaciones para responder de manera eficiente
int maxCiphers = (int) session.getAttribute("MaxExportCiphersPerPage");
//Variable que registra el total de filas que contiene el cruce de datos
int rowCount = conn.getInt("preCount");
//Variable que registra el total de columnas que contiene el cruce de datos
int columnCount = table.getLastColumnsChilds(table.getColumnsSortedByIndex(), new
ArrayList<AdvancedColumn>(), new HashSet<Integer>()).size();
//Total de cifras que conforman el ancho x alto que se obtuvieron de la consulta
int ciphers = rowCount * columnCount;
//Cantidad mínima de cifras que puede contener una página
int minCiphers = columnCount * defaultPageSize;
//Verifica si la cantidad actual de cifras del total de filas por el total de columnas sea
mayor que el mínimo requerido
if (minCiphers < ciphers) {
   //En caso afirmativo se incrementa la cantidad de filas que puede contener una página
   //sin sobrepasar maxCiphers (cantidad máxima de cifras por página)
   for (pageSize = defaultPageSize; ((pageSize * columnCount) < maxCiphers);
pageSize++) {
   }
```

```
    //ajusta el tamaño de página para evitar exceder el límite permitido por maxCiphers
    if ((pageSize * columnCount) > maxCiphers) {
        pageSize--;
    }
}
//Obtiene el total de páginas que contiene este cruce de datos
double pages = Math.ceil(rowCount / (double) pageSize);
...
```

6.6.7.3 Generación secuencial de páginas

Finalmente requerí la formación de cada una de las páginas que fueron calculadas en el paso anterior. Para cumplir con esta etapa solicité nuevamente el método **getAdvancedCrossTableTreeMeta()**, pero con un cambio en los parámetros para que ya no genere estructuras de columnas pero si de filas y cifras que son los restantes para formar un cruce de datos, a continuación se presenta el código que cumple con esta tarea:

```
//Se indica mediante este parámetro que se deben construir las estructuras de filas y las asignaciones de cifras al cruce de datos
conn.put("onlyColumns", false);
//Se inicializa la clase que se encarga de transformar las estructuras de datos en formato CSV
AdvancedCsvTransferExt advancedCsvTransferExt = new AdvancedCsvTransferExt();
//Ciclo que solicita una a una, las páginas que se requieren para formar la totalidad de información
for (int paging = 0; paging < pages; paging++) {
    //Parámetros requeridos para dar seguimiento a las páginas que son generadas
    conn.put("start", paging * pageSize);
    conn.put("page", paging + 1);
    conn.put("limit", pageSize);

    //Variable que recibe las estructuras de datos generadas
    table =
jsonConstructs.getAdvancedCrossTableTreeMeta(jsonConstructs.getQueryCross(conn),
jsonConstructs.getQueryAxisX(conn), jsonConstructs.getQueryAxisY(conn), conn,
oracleDataBaseManager, new Date().getTime());
    //Recuperación de columnas que son requeridas para el posicionamiento de las cifras
    table.setTableColumns(tableColumns);
    table.setPlainColumnTableByKey(plainColumnTableByKey);
    //Método que realiza la escritura de formato CSV en flujo de datos para respuesta a cliente
    advancedCsvTransferExt.getCsvTableFromData(table, csvWriter, paging == 0);
    //Liberación de flujo de datos
    csvWriter.flush();
    br.flush();
    //Solicitud de sistema para liberación de memoria por objetos no utilizados
```

```
    System.gc();
}
} catch (Exception ex) {
//Registro de errores
miscUtils.printSimpleError(ex, this.getClass().getName(), MiscUtils.ERROR);
} catch (Throwable thr) {
//Registro de errores
miscUtils.printSimpleError(thr, this.getClass().getName(), MiscUtils.ERROR);
} finally {
//En caso de éxito ó error se liberan y cierran los flujos de datos
try {
    csvWriter.flush();
    csvWriter.close();
    br.flush();
    br.close();
} catch (Exception e) {
}
//Asigna a la respuesta el tipo de salida, que en este caso es un flujo de datos
response = Response.ok(stream);
```

6.6.8 ExtJS Grid con BufferedStore (Servicio Web)

Es el servicio diseñado para responder el tipo de solicitudes que hace el componente **BufferedStore de Sencha Ext JS**, este componente de manera similar la formación de un archivo por páginas CSV, se crea a partir de la solicitud inicial de las columnas, y la solicitud secuencial de las páginas que conforman el cruce de datos final.

De esta manera quien vuelve a resolver las estructuras de datos es el método **getAdvancedCrossTableTreeMeta()**, realizando el mismo procesamiento pero la conversión final de las estructuras es en formato JSON, mismo que será recibido por el cliente para presentar los datos en forma de tabla, el componente cliente tiene la capacidad de consultar las páginas según sean requeridas por el cliente cuando se desplaza por los datos de manera vertical, estas consultas son automáticas y asíncronas, así se permite a la aplicación cliente lanzar solicitudes a los servicios web en procesos de fondo que evitan que se dificulte la navegación por la vista.

6.6.9 Cliente Web (Ext JS)

La interpretación de la respuesta es completamente tratada por el framework Ext JS 6.5.1 y el navegador web para ejecutar instrucciones JavaScript.

Ext JS sólo recibe parámetros y configuración para realizar las peticiones al WebService, atender la respuesta e interpretarla para visualizar alguno de sus componentes en el navegador Web.

La principal operación de comunicación con los Servicios Web, es por medio de solicitudes REST (Representational State Transfer) utilizando el componente Ext.Ajax de Ext JS, elementalmente están construidas de la siguiente manera:

```
Ext.Ajax.request({
    //Indicación de la ruta donde encontrar el Servicio Web
    url: 'resources/' + serviceName,
    //Tipo de petición Web
    method: 'POST',
    //Indicación de solicitud asíncrona
    async: async,
    //Función que actúa para atender la respuesta del Servicio Web
    success: function (response) {
        var jsonData = new Object();
        try {
            //Decodificación de texto a Objeto JavaScript
            jsonData = Ext.JSON.decode(response.responseText, false);
            if (jsonData.error === "") {
                var columnsTree = jsonData.columns;
                var dataTree = jsonData.data;
                dataTree.params = jsonData.params;
                var treeStore = Ext.create('Ext.data.TreeStore', dataTree);
                //Elemento principal encargado de reemplazar los datos anteriores por los
recibidos
                treepanel.reconfigure(treeStore, columnsTree);
            } else {
                //Elemento de reporte de error originado por el Servidor
                Ext.Msg.show({
                    title: 'Error Servidor',
                    msg: 'Error:' + jsonData.error,
                    icon: Ext.Msg.ERROR,
                    closable: false,
                    buttons: Ext.Msg.OK
                });
            }
        } catch (e) {
            //Elemento de reporte de error originado por el cliente
            Ext.Msg.show({
                title: 'Error Cliente',
                msg: 'Error:' + e,
                icon: Ext.Msg.ERROR,
                closable: false,
                buttons: Ext.Msg.YES,
                fn: function (btn, text) {
                }
            });
        }
    },
    //Elemento por el cual se envía la parametría al Servicio Web
    jsonData: params});
```

Esta implementación de código en JavaScript, es la operación más recurrente, pues es la forma en la que se reciben las estructuras destinadas a un tipo TreePanel de Ext JS.

6.6.10 ExtJS Grid con BufferedStore

Este componente permitirá descargar una cantidad sin fin de datos debido a que trabaja con transacciones que son almacenadas en un búfer conforme son solicitadas por el cliente.

Esta herramienta del Framework Ext JS, permite la presentación de datos en forma de tabla, con un crecimiento indefinido para el eje vertical, no así para el eje horizontal de la tabla, pues al incrementar la cantidad de datos horizontales, el componente gráfico comienza a mostrar mucha lentitud e incrementar en gran medida la demanda de memoria ram y procesador por parte del navegador web.

La vista final de un llenado de datos utilizando este componente, es la siguiente:

	Año	Fracción Ar...	Pesos	Fletes y S...
1	2008	ANIMALES...	0.000	
2	2003	ANIMALES...	6,194.000	0.040
3	2004	ANIMALES...	7,479.000	0.040
4	2005	ANIMALES...	2,973.000	0.040
5	2006	ANIMALES...	507.000	0.040
6	2007	ANIMALES...	1,101.000	0.040
7	2008	ANIMALES...	1,619.000	0.040
8	2009	ANIMALES...	1,516.000	0.040
9	2010	ANIMALES...	576.000	0.040
10	2011	ANIMALES...	752.000	0.040
11	2012	ANIMALES...	1,052.000	0.040
12	2003	ANIMALES...	736.000	0.080
13	2004	ANIMALES...	70.000	0.080
14	2005	ANIMALES...	79.000	0.080
15	2006	ANIMALES...	0.000	0.080
16	2007	ANIMALES...	41.000	0.080
17	2008	ANIMALES...	226.000	0.080
18	2009	ANIMALES...	132.000	0.080
19	2010	ANIMALES...	9.000	0.080
20	2011	ANIMALES...	28.000	0.080
21	2013	ANIMALES...	3,995.000	
22	2014	ANIMALES...	5,227.000	
23	2012	ANIMALES...	49,368.000	
24	2013	ANIMALES...	247,375.000	

Pagina 1 de 27 Mostrando 1 - 6250 de 167351

Figura 28. Vista web con BufferedStore

En esta vista se observan las columnas y filas generadas, y en la parte inferior se muestra el conteo de páginas, para el ejemplo se muestra la primer página de 27 totales. Hablando de registros se muestran registros o filas que conforman la página, 6250 filas están contenidas en una página, esto de un total de 167,361 registros o filas totales.

El proceso general de la formación de una tabla con estas características, está descrita en el siguiente diagrama:

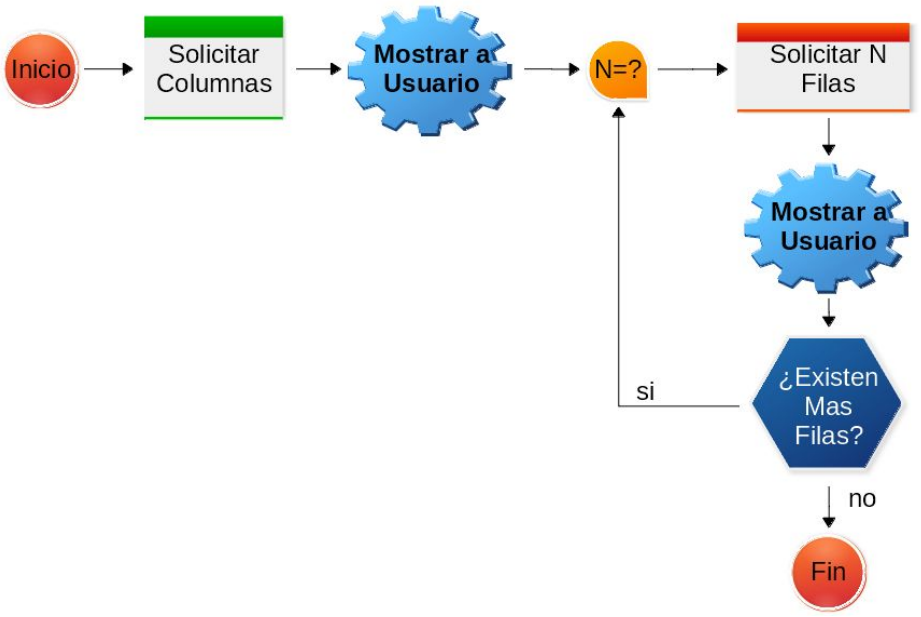

Figura 29. Interacción de BufferedStore con WebService

6.6.10.1 Calcular segmentación de información (Paginación)

De igual manera como se realizó el cálculo en el servicio web, se realiza en el cliente a fin de asegurar un buen funcionamiento de la aplicación, además de asegurar las solicitudes provenientes del servicio web, donde un usuario podría saturar el servidor con la alteración de los parámetros calculados y así solicitar la generación de páginas de gran tamaño, esto ocasionará el colapso del servicio.

El código correspondiente en JavaScript es el siguiente:

```
//Se inicializa el valor por defecto de un número filas por página, en este caso 25
var pageSize = defaultPageSize = 25;
//Se recupera el valor proveniente de WebService que indica el máximo número de cifras
que puede contener una página
var maxCiphers = TrSConexion.maxCiphersPerPage;
//Se multiplica el ancho x alto de los datos que se tienen, esto da la cantidad total de
cifras por el cruce de datos
var ciphers = dimensions.A * dimensions.B;
//Se calcula el producto de las columnas por el valor default de filas por página
//este valor nos da la cantidad mínima de cifras que pude contener nuestro cruce de datos
var minCiphers = (params.transpose ? dimensions.A : dimensions.B) *
defaultPageSize;
//Verifica si la cantidad actual de cifras del total de filas por el total de columnas sea
```

```
mayor que el mínimo requerido
if (minCiphers < ciphers) {
   //En caso afirmativo se incrementa la cantidad de filas que puede contener una página
   //sin sobrepasar maxCiphers (cantidad máxima de cifras por página)
   if (Math.ceil(ciphers / minCiphers) > 3) {
      for (pageSize = defaultPageSize; ((pageSize * (params.transpose ? dimensions.A :
dimensions.B)) < maxCiphers) && (Math.ceil(ciphers / (pageSize * (params.transpose ?
dimensions.A : dimensions.B))) > 3); pageSize++) {
      }
      ;
   }

   //Ajusta el tamaño de página para evitar exceder el límite permitido por maxCiphers
   if ((pageSize * (params.transpose ? dimensions.A : dimensions.B)) > maxCiphers) {
      pageSize--;
   }
}

//Obtiene la cantidad de páginas mínimas que se deben conservar
var leadingBufferZone = pageSize * 2;
...
```

6.6.10.2 BufferedStore

Es el objeto responsable de formar esta vista de tabla, es Ext.data.BufferedStore proveniente del conjunto de elementos de Sencha Ext JS.

En la Figura 34 se ilustra la solicitud inicial de columnas, estas son mostradas en la tabla para posteriormente enviar la solicitud de N cantidad de páginas iniciales para ser mostradas en la interfaz del usuario, N está predefinida en la definición del componente.

Una vez que se ha calculado la paginación que tendrá esta vista, se puede crear el componente, no es posible hacerlo antes pues no hay forma de modificar estos valores en tiempo de ejecución, así que es necesario conocer ya los valores.

El siguiente código ilustra la creación del componente:

```
//Creación de elemento
Ext.create('Ext.data.BufferedStore', {
   //Indica que debe activar el almacenamiento de páginas disponibles
   buffered: true,
   //Indica que debe lanzar las nuevas solicitudes si las páginas vistas se acercan a las
almacenadas
   autoLoad: true,
   //Indica que debe ejecutar la primera carga en la creación del componente
   firstLoad: true,
```

```
//Indica la cantidad de páginas de reserva del componente
leadingBufferZone: leadingBufferZone,
//Cantidad de filas por página
pageSize: pageSize,
//Objeto encargado de lanzar las solicitudes al WebService
proxy: {
    //Tipo de solicitud a WebService, en este caso REST
    type: 'rest',
    //Ruta hacia el servicio web
    url: 'resources/tableCrossService',
    //Límite de tiempo en que se debe esperar por una solicitud por parte del servidor
    timeout: 9000000,
    //Objeto lector de los datos de entrada
    reader: {
        //Posiciona los datos recibidos en la estructura cliente
        rootProperty: 'topics',
        totalProperty: 'totalCount',
        //Elemento que puede manipular los datos recibidos
        transform: function (data) {
            //Carga de Columnas de la primer solicitud
            if (data.hasOwnProperty('columns')) {
                grid.reconfigure(data.columns);
                config.fields = data.fields;
                grid.preCount = data.preCount;
            } else if (data.hasOwnProperty('error')) {
                var loadMask = new Ext.LoadMask({
                    msg: loadMsg,
                    target: grid
                });
                loadMask.show();
                Ext.Msg.show({
                    title: 'Error en Servidor',
                    msg: 'Error:' + data.error,
                    icon: Ext.Msg.ERROR,
                    closable: false,
                    buttons: Ext.Msg.OK,
                    fn: function (btn, text) {
                        loadMask.destroy();
                        if (data.error === 'Sesión Inválida') {
                            window.location.reload();
                        }
                    }
                });
            }
            return data;
        }
    },
    //Indicación de generar sólo columnas
    extraParams: {
        onlyColumns: true,
```

```
            queryParams: Ext.JSON.encode(params)
        }
    },
    listeners: {
        load: function (store, records, successful, operation, eOpts) {
            //Asignación de valores para la primer solicitud
            if (store.firstLoad) {
                loadMask.destroy();
                store.getProxy().setExtraParams({
                    onlyColumns: false,
                    preCount: grid.preCount,
                    columnCount: params.transpose ? dimensions.A : dimensions.B,
                    queryParams: Ext.JSON.encode(params)
                });
                store.setFields(config.fields);
                grid.reconfigure(store);
                store.firstLoad = false;
                store.read();
            } else {
                //Asignación de valores para la segunda y subsecuentes solicitudes.
                grid.totalPages = Math.ceil(store.totalCount / store.pageSize);
                grid.deltaScroll = Math.ceil(grid.getScrollable().getMaxPosition().y /
grid.totalPages);
                grid.dockedItems.getByKey('toolbarPaging').update(store);
            }
        }
    }
});
```

Uno de los elementos más importantes de este elemento es la definición de la propiedad **leadingBufferZone**, la cual define gran parte del comportamiento del componente.

La propiedad indica que la aplicación debe almacenar en todo momento el valor numérico de esta propiedad en páginas, es decir si su valor es de 3, obliga al componente a solicitar inicialmente 3 páginas al Servicio Web y mantener ese número de reserva cada que el usuario se desplace por la tabla generada.

Si el usuario se desplaza a la segunda página, en automático se solicita una nueva página al WebService para nuevamente tener 3 páginas de reserva, así se garantiza que el usuario visualice en todo momento una cierta cantidad de información sin esperar a que sean solicitados los datos, hace que la experiencia de usuario sea muy fluida y satisfactoria al brindarle el efecto de tener la totalidad de los datos en su pantalla.

6.6.11 Respuesta por Flujos

Para las ya mencionadas estructuras arbóreas como Índice, Eje Vertical o Eje Horizontal, realicé la implementación de una respuesta que libera el flujo de datos de forma continua con el fin de no saturar la memoria del servidor de aplicaciones y al mismo tiempo acelerar las transacciones.

El número de usuarios ha incrementado, calculé una concurrencia de hasta 80 usuarios consultando grandes cantidades de información por lo que hice cambios para solventar la cantidad de solicitudes a la aplicación.

Los dos tipos de respuestas por Flujos que utilicé en este trabajo son las siguientes:

```
//Para cuando se va a responder al cliente con un flujo de Bytes (XLSX)
response = Response.ok(new ByteArrayOutputStream());
//Para cuando se va a responder al cliente con un texto, ya sea JSON o CSV
response = Response.ok(new StreamingOutput());
```

7 Conclusiones

En la realización de los proyectos expuestos en este trabajo pude aplicar los conocimientos tecnológicos adquiridos, donde las principales aplicaciones de estos conocimientos se reflejan en la mejora de los procesos de información para la generación de índices económicos del Instituto.

Me fue posible tomar los recursos con los que contaba el Instituto y hacer uso de ellos a favor de los diferentes proyectos dedicados a sistematización. Observé la capacidad con la que contaban pues los equipos y sus alcances estaban sobrepasados tomando en cuenta las tareas que llevaban a cabo, con ello alojé herramientas que potenciaron la velocidad y cantidad de procesamiento de información que cada vez era más demandante.

Uno de los recursos más aprovechados por los proyectos aquí presentados es el uso de la red de datos, con capacidad suficiente para soportar las transacciones habituales y una gran carga de nuevos intercambios de información, característica que fue indispensable para llevar información de manera eficiente a las bases de datos centralizadas en el estado de Aguascalientes.

Fue posible la aplicación de los desarrollos existentes para el impulso que debía tomar cada proyecto y poder salir a un ambiente productivo, en gran medida el uso de la correcta tecnología marcó la diferencia en el éxito o no de un producto.

Gracias a la flexibilidad del diseño de los proyectos generé múltiples propuestas entre las que siempre se consideraron implementación de herramientas existentes o en su defecto el desarrollo de nuevas, además de procedimientos que fortalezcan las actuales para la solvencia a problemáticas que de forma recurrente se relacionaban con la capacidad de tratar información y almacenarla para su posterior utilización.

También mezclé tecnologías de origen libre, privativo y desarrollos propios para un fin común, donde las bases de datos, servidores de aplicaciones y framework Extjs están bajo licenciamiento, y Java como plataforma de desarrollo, para generación de código libre.

La sistematización de un procedimiento fue la motivación de cada una de las herramientas presentadas, donde un proceso hecho por un usuario podría ser traducido a una serie de instrucciones ejecutadas por un equipo informático que lo repitiera con menor lugar a fallos y tiempos de procesamiento, aumentando así la productividad de los usuarios y mejorando su capacidad de análisis y visualización de los datos.

Uno de los rubros más importantes en los proyectos presentados es el volumen de información que el Instituto maneja, las herramientas convencionales están diseñadas para trabajar con ambientes comunes donde no se consideran el paso de los límites establecidos. Los desarrollos de este trabajo fueron diseñados para dar soporte a la

cantidad de datos que se recibía y también fueron preparadas para soportar los futuros incrementos que los datos puedan tener.

Otra de las características de la información manipulada es la diversidad en estructura que se presentó, al no existir un modelo de datos estándar, diseñé un modelo que pudiera almacenar información de todo tipo, bajo las reglas y significados con que fue generada originalmente, es decir sin pérdida en cualquiera de los atributos que den identidad a los datos, así este modelo al ser flexible cumple con las capacidades necesarias de almacén y explotación de información, abriendo el camino a la incorporación de diversas herramientas o procedimientos que dan mayor significado al conjunto de datos por el hecho de mostrar indicadores de su comportamiento.

Las herramientas aquí mostradas, han sido sometidas a pruebas de estrés, de calidad y de usuario; pasando por estas pruebas actualmente se encuentran en estado productivo.

8 Bibliografía

Point, Tutorials. “Java Overview.” *www.tutorialspoint.com*. Tutorials Point, 2017. Web. 15 Sept. 2017. <https://www.tutorialspoint.com/java/java_overview.htm>

Point, Tutorials. “Java Networking.” *www.tutorialspoint.com*. Tutorials Point, 2017. Web. 15 Sept. 2017. <https://www.tutorialspoint.com/java/java_networking.htm>

What Are Web Services? - The Java EE 6 Tutorial. N.p., Jan. 2013. Web. 15 Sept. 2017. <http://docs.oracle.com/javaee/6/tutorial/doc/gijvh.html>

Building Web Services with JAX-WS - The Java EE 6 Tutorial. N.p., Jan. 2013. Web. 15 Sept. 2017. <http://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html>

Types of Web Services - The Java EE 6 Tutorial. N.p., Jan. 2013. Web. 15 Sept. 2017. <http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html>

“Processes and Threads.” *Processes and Threads (The Java™ Tutorials > Essential Classes > Concurrency)*. N.p., n.d. Web. 15 Sept. 2017. <https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html>

Thread (Java Platform SE 7). N.p., Nov. 2016. Web. 15 Sept. 2017. <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>

“Java ArrayList Class - Javatpoint.” *www.javatpoint.com*. N.p., n.d. Web. 15 Sept. 2017. <https://www.javatpoint.com/java-arraylist>

“HashMap in Java - Javatpoint.” *www.javatpoint.com*. N.p., n.d. Web. 15 Sept. 2017. <https://www.javatpoint.com/java-hashmap>

org.json: public class: JSONObject. N.p., n.d. Web. 15 Sept. 2017. <http://www.docjar.com/docs/api/org/json/JSONObject.html>

BillWagner. “Introduction to the C# Language and the .NET Framework.” *Microsoft Docs*. N.p., n.d. Web. 15 Sept. 2017.

<https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>

JavaScript Tutorial. N.p., n.d. Web. 15 Sept. 2017. <https://www.w3schools.com/js/default.asp>

JavaScript Objects. N.p., n.d. Web. 15 Sept. 2017. <https://www.w3schools.com/js/js_objects.asp>

JavaScript Arrays. N.p., n.d. Web. 15 Sept. 2017. <https://www.w3schools.com/js/js_arrays.asp>

JavaScript Functions. N.p., n.d. Web. 15 Sept. 2017. <https://www.w3schools.com/js/js_functions.asp>

AJAX Introduction. N.p., n.d. Web. 15 Sept. 2017. <https://www.w3schools.com/xml/ajax_intro.asp>

"Sencha Ext JS." *Sencha.com*. N.p., Jan. 2017. Web. 15 Sept. 2017. <https://www.sencha.com/products/extjs/#overview>

"Ext.data.Store | Ext JS 6.5.1." *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.Store.html>

"Ext.data.Model | Ext JS 6.5.1." *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.Model.html>

"Ext.data.proxy.Proxy | Ext JS 6.5.1." *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.proxy.Proxy.html>

"Ext.data.BufferedStore | Ext JS 6.5.1." *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.BufferedStore.html>

Grid with Buffered Store. N.p., n.d. Web. 15 Sept. 2017. <http://examples.sencha.com/extjs/6.5.1/examples/classic/grid/buffered-store.html>

"Ext.tree.Panel | Ext JS 6.5.1." *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.tree.Panel.html>

“Ext.data.TreeStore | Ext JS 6.5.1.” *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.TreeStore.html>

“Ext.data.proxy.Ajax | Ext JS 6.5.1.” *Sencha Documentation*. N.p., n.d. Web. 15 Sept. 2017. <http://docs.sencha.com/extjs/6.5.1/classic/Ext.data.proxy.Ajax.html>

“The Java API for Microsoft Documents.” *Apache POI*. N.p., n.d. Web. 15 Sept. 2017. <https://poi.apache.org/>

Welcome to The Apache Software Foundation! N.p., n.d. Web. 15 Sept. 2017. <https://www.apache.org/licenses/LICENSE-2.0>

“Database Utilities.” *SQL*Loader Concepts*. N.p., Apr. 2017. Web. 15 Sept. 2017. <https://docs.oracle.com/database/122/SUTIL/oracle-sql-loader-concepts.htm#SUTIL003>

Acerca de los cubos OLAP. N.p., n.d. Web. 15 Sept. 2017. <https://msdn.microsoft.com/es-es/library/hh916536(v=sc.12).aspx>