PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

Análisis y Diseño de una herramienta de desarrollo de soluciones para Inteligencia de Negocios

Módulo de Extracción

Tesis para optar por el Título de Ingeniero Informático

Presentada por:

Pilar Eliana Infantas Asunción

César Manuel Mendoza Suárez

María Magdalena Uribe Uribe

LIMA – PERÚ

2007

RESUMEN

La Tecnología de Información (TI) es, en la actualidad, un componente de gran importancia para cualquier organización. Sin embargo, son los datos y su adecuado manejo como transformaciones, búsqueda de patrones, y consolidaciones; lo que le da un carácter estratégico a la TI en la organización. En este contexto es donde aparecen conceptos como el de Inteligencia de Negocios, que apoyados en técnicas, estrategias, metodologías y herramientas buscan ofrecer información más adecuada para la toma de decisiones. Una solución de Inteligencia de Negocios puede, con gran posibilidad, cambiar el rumbo de una organización.

La implementación de soluciones de Inteligencia de Negocios se apoya necesariamente en un conjunto de herramientas informáticas que tienen que cubrir un ciclo de trabajo que comienza con la definición de un almacén de datos o Data Warehouse, la extracción y transformación de los datos desde diversas fuentes de información, y finalmente, la explotación de la información a través de diversos reportes tabulares y gráficos que permitan a la alta dirección de una organización la toma de decisiones.

El presente proyecto de tesis busca realizar el análisis y diseño del módulo de extracción de una herramienta básica para Soluciones de Inteligencia de Negocios que cubra todos los procesos del ciclo de trabajo. La arquitectura permitirá que una organización provea el servicio de Inteligencia de Negocios a múltiples organizaciones. Además, se toma en cuenta la escalabilidad del producto para soportar mayor número de fuentes de datos en futuras versiones.

FACULTAD
DE INGENIERÍA
Especialidad de Ingeniería Informática

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
90 AÑOS

TEMA DE TESIS PARA OPTAR EL TÍTULO DE INGENIERO INFORMÁTICO

TÍTULO: **"Análisis y Diseño de una herramienta de desarrollo de soluciones para inteligencia de negocios – Módulo de Extracción".**

ÁREA: Sistemas de Información

ASESOR: Ing. Luis Morales.
Ing. Carla Basurto
Ing. Luis Flores
Ing. Abraham Dávila

ESTUDIANTES	**CODIGO**	**NOMBRES**
	2000.0423.2.12	Pilar Eliana Infantas Asunción
	2000.7175.5.12	María Magdalena Uribe Uribe
	2000.7306.7.12	César Manuel Mendoza Suárez

TEMA N°: 208

FECHA: Lima, 04 de diciembre de 2006

DESCRIPCIÓN

La Tecnología de Información (TI) es, en la actualidad, un componente de gran importancia para cualquier organización, sin embargo, son los datos y su adecuado manejo como transformaciones, búsqueda de patrones, y consolidaciones; lo que le da un carácter estratégico a TI en la organización. En este contexto es donde aparecen conceptos, como el de Inteligencia de Negocios que apoyados en técnicas, estrategias, metodologías y herramientas buscan ofrecer información más adecuada para la toma de decisiones. Una decisión adecuada en una organización se traduce en mejoras significativas sea en dinero u otro beneficio. Una solución de Inteligencia de Negocios puede con gran posibilidad, cambiar el rumbo de una organización hacia escenarios más favorables y más beneficiosos.

En el mercado existen diversas herramientas que apoyan la implementación de Inteligencia de Negocios, pero son muy pocas las organizaciones que los utilizan en nuestro país, principalmente por el alto costo que implican la plataforma informática de este tipo de soluciones o porque las soluciones existentes cubren una parte y no todo el espectro de posibilidades.

La implementación de soluciones de Inteligencia de Negocios se apoya necesariamente en un conjunto de herramientas informáticas que tienen que cubrir un ciclo de trabajo que comienza con la extracción de los datos desde diversas fuentes de información como archivos de bases de datos de diferentes proveedores, hojas de cálculo, archivos planos, entre otros; continuando con un proceso de transformación de los mismos que puede ser tan simple como la homogeneización de los datos a conversiones complejas que se realizan en varias etapas; y, finalmente el análisis de información a través de diversos reportes tabulares y gráficos que permitan a la alta dirección de una organización tomar decisiones. Una solución básica de inteligencia

de negocios utiliza grandes almacenes de datos (*data warehouse*) y herramientas que ayuden al diseño de las transformaciones y la explotación de la información; cada cual con sus diversas complejidades.

El presente proyecto de tesis busca implementar una solución que permita el desarrollo de una herramienta básica para Soluciones de Inteligencia de Negocios. La arquitectura permitirá que una organización provea el servicio de Inteligencia de Negocios, en vez de pensar en un producto a ser vendido; por lo que será necesario definir una estrategia para el manejo de este servicio dentro de la solución que se va a implementar; asimismo esto implica que la solución debe ser escalable tanto a nivel de nuevo hardware a ser incorporado como inclusión de más componentes en la solución como parte de la evaluación misma del producto.

OBJETIVOS Y ALCANCE

Desarrollar una Herramienta Integral que permita desarrollar y explotar el Data Warehouse de una empresa o de varias de ellas a través del Internet.

La solución se construirá basada en la especificación J2EE e incorpora los siguientes módulos:

- Análisis Dimensional. Permite la administración del modelo estrella de un Data Warehouse.
- Extracción. Permite la administración y ejecución de los componentes que capturan los datos desde su origen hasta llevarlos al repositorio de Data Warehouse respectivo.
- Explotación: Permite la administración de los componentes que van a permitir visualizar la información del Data Warehouse.

Este proyecto de tesis cubrirá el Análisis y Diseño del módulo de Extracción, el cual cuenta con los siguientes sub-módulos:

- **Administración de Paquetes:** Permite la creación de paquetes para realizar la Extracción, Transformación y Carga de datos hacia el Data Warehouse.

- **Visualización de Componentes:** Permite visualizar de una manera amigable los componentes creados y las reglas de carga definidas por el usuario.

- **Extracción:** Permite realizar la conexión a los datos fuentes, el mapeo de los datos, la estandarización y el filtro de dichos datos, y la validación de dicha manipulación de datos.

- **Transformación y Carga:** Permite realizar la personalización de la manipulación de los datos extraídos y cargar dichos datos hacia el Data Warehouse. Dicha personalización también puede involucrar funciones predefinidas por el usuario.

- **Administración Jobs:** Permite la administración, control y supervisión de los procesos de ejecución de los diversos paquetes creados en el proceso de extracción.

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
Facultad de Ciencias e Ingeniería
14 FEB. 2007
EDUARDO ISMODES CASCÓN
DECANO

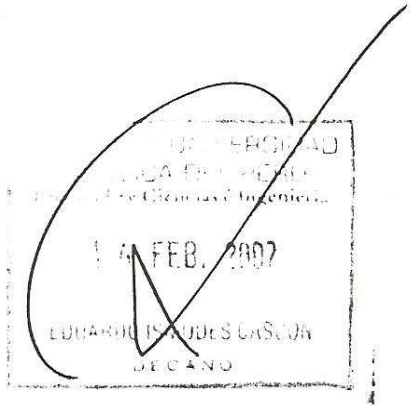
Adicionalmente, este módulo forma parte de la arquitectura integral de la herramienta de Inteligencia de Negocios.

Finalmente, se han desarrollado estos módulos de la siguiente manera:

SUBMÓDULOS	RESPONSABLES
Administración Paquetes	César Manuel Mendoza Suárez
Visualización de Componentes	Pilar Eliana Infantas Asunción
Extracción	Pilar Eliana Infantas Asunción
	María Magdalena Uribe Uribe
Transformación y Carga	María Magdalena Uribe Uribe
	César Manuel Mendoza Suárez
Administración Jobs	María Magdalena Uribe Uribe

Este documento de tesis se complementa con la tesis Construcción y Pruebas de una herramienta de desarrollo de soluciones para inteligencia de negocios – Módulo de extracción. El equipo de trabajo fue uno sólo y cubrieron todas las actividades necesarias para la operatividad del producto.

Máximo: 100 páginas



"A mis padres, por su inmenso amor, apoyo y constante guía en cada paso de mi vida. A ti, porque aún ausente estás cada día en mi corazón."

-Pilar Infantas

"A mi papá Julio César, por su ejemplo y los sabios consejos, y a mi mamá Benita, por su inmenso amor y alegría; porque ambos son una inspiración para mi."

-César Mendoza

“A mi familia, por guiarme a lo largo de estos años, por ayudarme a superar todos los obstáculos que se presentaron, y sobretodo por creer en mi y amarme en todo momento.”

-María Uribe

De manera muy especial agradecemos, a la Pontificia Universidad Católica del Perú por la formación integral que nos brindó.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad, la Inteligencia de Negocios es dentro de las organizaciones una pieza clave para una adecuada y oportuna toma de decisiones. Una solución de Inteligencia de Negocios puede cambiar el rumbo de una organización hacia escenarios más favorables y más beneficiosos.

La herramienta propuesta es una herramienta integral que permite desarrollar y explotar el *Data Warehouse* de una empresa o de varias de ellas a través de Internet. Se desarrolló el módulo de Extracción, cuyo objetivo es el de permitir la administración y ejecución de los componentes que capturan los datos desde su origen hasta llevarlos al repositorio de *Data Warehouse* respectivo. Este módulo forma parte de la arquitectura integral de la herramienta de Inteligencia de Negocios, la cual consiste en tres módulos con funciones específicas. Estos módulos son: Análisis, Extracción y Explotación.

El módulo de Análisis es el que inicia el flujo de trabajo con la herramienta y define las estructuras de datos para el *Data Warehouse* en el que se colocarán los datos procesados.

El módulo de Extracción es el que se encarga de realizar los procesos de extracción, transformación y carga de datos, desde las fuentes de datos de la organización hacia el *Data Warehouse*.

El módulo de Explotación es el que se encarga de completar el flujo de trabajo, explotando los datos del *Data Warehouse* y mostrando reportes con la información requerida por el usuario final.

A lo largo del presente documento de tesis se describe el estudio realizado para el análisis y diseño del módulo de extracción de la herramienta propuesta. Es importante mencionar que este trabajo de tesis es el punto de partida para la tesis: “Construcción y

Pruebas de una herramienta de desarrollo de soluciones para inteligencia de negocios – Módulo de Extracción", realizada por Luis Dall'Orto y David Wu.

En el primer capítulo, se presenta un análisis a la problemática existente y en base a la misma se plantea la solución. En este punto se considera importante comparar la herramienta creada con las ofrecidas actualmente en el mercado.

En el segundo capítulo, se presentan las características del entorno de desarrollo y de pruebas de la herramienta, así como también los perfiles de usuarios del sistema. Se presentan los casos de uso que se desarrollaron para satisfacer los requerimientos planteados para la herramienta.

Seguidamente, en el tercer capítulo, se muestra el diseño completo realizado utilizando el lenguaje UML como lenguaje para la construcción de artefactos de *software*. Los siguientes artefactos fueron desarrollados: diagramas de casos de uso, diagramas de secuencias, diagrama de arquitectura, diagrama de clases. En este capítulo se presenta también la descripción de los principales algoritmos usados en la herramienta, así como el diseño de las pantallas usadas.

Finalmente en el cuarto capítulo, se presentan las observaciones, conclusiones y recomendaciones pertinentes.

1. MARCO TEÓRICO

1.1. Definición de la problemática

Actualmente la Tecnología de la Información se ha convertido en una herramienta clave en el proceso de desarrollo continuo dentro de las empresas. La competitividad del mercado y la globalización de la industria plantean un reto mayor dentro de toda organización pues se hace necesaria la innovación y el planeamiento estratégico que permita a la empresa trascender con un producto o servicio diferenciado.

Un componente indispensable en la toma de decisiones es el manejo eficaz y eficiente de los datos y la información que forma parte del conocimiento de la organización. En este contexto es donde aparecen conceptos como el de Inteligencia de Negocios que buscan ofrecer los resultados más adecuados para las organizaciones.

Una empresa que no sepa adaptar los nuevos conceptos de Inteligencia de Negocios en sus procesos de toma de decisión y manejo eficiente de la información, cae en el riesgo de quedar rezagada frente a un mercado cada vez más competitivo, que sabe aprovechar de buena forma los recursos tecnológicos y las tecnologías de información.

Uno de los activos valiosos que cada vez más se explotan en toda empresa es la información y los datos propios del negocio, almacenados durante el tiempo de vida de una organización. Sin importar el rubro en el cual se desarrolle, toda empresa genera durante sus operaciones diarias una cantidad de datos que se guardan en diferentes de formas (desde bases de datos especializadas de distintos proveedores hasta archivos en formatos de hoja de cálculo). El problema es que no siempre esta información está siendo explotada de forma inteligente. Lo óptimo es poder incorporar todos esos resultados en la toma de decisiones futuras, y así formar un planeamiento más real. Todo negocio debe siempre estar aprendiendo

de sí mismo.

El manejo de estos datos se sumerge ahora en toda una nueva forma de uso de la información, que requiere de un análisis no convencional. El hecho de implantar estas nuevas herramientas que permiten sacar provecho a los datos del negocio, es actualmente una labor complicada. Inicialmente, al no contar con una forma automatizada de lograr esa incorporación, eran necesarios muchos recursos, un análisis exhaustivo y migraciones complejas.

Actualmente, en el mercado existen algunas herramientas que apoyan la implementación de soluciones de Inteligencia de Negocios, pero son muy pocas las organizaciones que los utilizan en nuestro país, principalmente por el alto costo que implica implantar la plataforma informática de este tipo de soluciones o porque las soluciones existentes no se adaptan al espectro de posibilidades que las empresas necesitan.

Para tener una mejor visión del alcance y lo que implica la elaboración de esta herramienta de Inteligencia de Negocios, presentamos la definición de los principales conceptos que están comprometidos con este tema.

a) Inteligencia de Negocios

Según Almeida [ALM 1999], se la puede definir como el uso de los datos recopilados con el fin de generar mejores decisiones de negocio, esto implica accesibilidad, análisis y revelar nuevas oportunidades.

Algunos conceptos de inteligencia de negocios no son nuevos, pero incluyen ahora la experiencia ganada desde los sistemas de información centrales hasta las aplicaciones de *data warehouse*.

La inteligencia de negocios busca proveer de un conjunto de tecnologías y productos para proporcionar a los usuarios la información que necesitan para resolver preguntas de negocios y tomar decisiones tácticas y estratégicas para el negocio.

b) Data Warehouse

Kimball [KIM 2002] lo define como la conglomeración de un conjunto de datos, los cuales se requieren almacenar y presentar de forma organizada. Los datos de la operación que se almacenan están estructurados de tal forma que puedan ser consultados con el fin de analizarlos.

Según Kimball, podemos distinguir algunos elementos básicos del *Data Warehouse*:

- Sistemas de Fuente Operacionales, la arquitectura en la cual se almacena los datos de la operación de la empresa.
- Área de arreglo de los Datos, donde se ejecutan la depuración, estandarización y combinación de los datos de la fuente operacional. Además se almacena los datos

y se ejecuta procesos de ordenamiento.

- Área de Presentación de los Datos, donde se realiza la carga de los datos que conforman un *Data Mart*, haciendo uso del modelamiento dimensional.
- Herramientas de Acceso a los Datos, que incluye aplicaciones para consulta específica, generadores de reportes, análisis de datos, modelamiento de proyecciones y estimación de resultados.

c) Data Mart

Kimball lo define como subconjunto lógico y físico del área de presentación de datos en un *Data Warehouse*. Originalmente, los *data mart* fueron definidos como un subconjunto altamente agregado de datos, normalmente usados para resolver preguntas específicas del negocio. Esta definición resultó no ser la más apropiada pues provocaba que los *data mart* sean inflexibles de combinarse con otros.

Esta primera concepción ha sido reemplazada, y el *data mart* es ahora definido como un conjunto flexible de datos, idealmente basado en los datos más atómicos posibles que se puedan extraer de una fuente operacional, y presentados en un modelo dimensional que es el que posee mayor capacidad de recuperación ante consultas inesperadas de los usuarios.

Los *data mart* pueden estar vinculados usando técnicas específicas al momento de conformar sus dimensiones. En este caso decimos que los *data mart* están conectados al bus del *data warehouse*.

En una forma simplificada, podemos decir que un *data mart* representa los datos de un proceso único de negocio.

d) Data Mining

Es una clase de consultas indirectas, normalmente sobre la data más atómica, que busca encontrar patrones inesperados en los datos. Los resultados más valiosos del *data mining* se obtienen agrupando, clasificando, estimando, prediciendo y encontrando acciones que ocurren juntas. Hay muchos tipos de herramientas que participan dentro del *data mining*.

La principal herramienta puede incluir árboles de decisiones, redes neuronales, herramientas de razonamiento basado en casos, herramientas de visualización, algoritmos genéticos y estadística clásica. Generalmente el *data mining* es un usuario del *data warehouse*.

e) Modelamiento Dimensional

Según Kimball, constituye una forma de modelamiento lógico de los datos orientado al rendimiento de las consultas y la facilidad de uso que se inicia de un conjunto de eventos de mediciones básicas. En el ámbito del modelo relacional de base de datos, una tabla *Fact* está acompañada de un conjunto de tablas de dimensión que le describen los atributos en el contexto de cada registro medidor. Por su estructura característica, al modelo dimensional se le conoce también como modelo de esquema estrella.

Un modelo dimensional busca con su diseño proveer claridad, predicción, escalabilidad y alta resistencia a una significativa cantidad de consultas, todo ello debido a su naturaleza simétrica. Dicho modelos dimensionales son la base de muchos componentes que agregan rendimiento en las bases de datos, incluyendo su considerable facilidad para poder vincular diferentes jerarquías de datos y aproximación por índices.

Los modelos dimensionales son la base para el desarrollo incremental y distribuido de los *data warehouse* a través del uso de dimensiones y Facts, además son la base lógica para los sistemas OLAP.

f) Fact Table / Tabla Fact

Kimball define que en un esquema tipo estrella (modelo dimensional), la tabla *Fact* representa la tabla central con medidores numéricos de rendimiento caracterizadas por una llave compuesta, cada una de ellas es una llave foránea en las tablas de dimensión.

g) Tabla de dimensión

Kimball define que en el modelo dimensional, una tabla de dimensión es una tabla con una única llave primaria y varias columnas de atributos descriptivos.

h) ETL (Extracción-transformación-carga)

Según Kimball el ETL es un conjunto de procesos por medio de los cuales los datos de la fuente operacional son preparados para colocarse en el *data warehouse*. El proceso primario de la preparación de los datos en el área de arreglo de datos de un *data warehouse*, antes de la presentación y la consulta.

El ETL consiste en extraer los datos de la fuente de origen, transformarla, cargarla e indexarla, asegurando su integridad, coherencia y disponibilidad en el destino.

i) Medida

Kimball lo define como una medición de rendimiento del negocio, típicamente numérico y aditivo, que es guardado en una Tabla Fact.

1.2. Objetivo principal

El objetivo principal de esta parte del proyecto es realizar el análisis y diseño de una herramienta ETL para su posterior construcción, la cual permita conectar el flujo de trabajo entre las otras dos partes del proyecto, el módulo de Análisis y el módulo de Explotación. El flujo de trabajo es el siguiente: luego de que se haya definido en el módulo de Análisis el esquema del *data mart* en el cual se almacenarán los datos históricos, se hace uso del módulo de Extracción para construir los flujos de transformación que cargarán los datos en el *data mart*. Finalmente, luego de haber ejecutado la carga de datos, mediante el módulo de Explotación se podrá obtener los reportes que mostrarán la información buscada.

Es importante notar que los datos de origen que serán cargados en el *data mart* pueden provenir de fuentes de datos heterogéneas. Esto hace que sea necesario pensar en una solución ETL que soporte el espectro de fuentes de datos existentes en el mercado en la actualidad, pero que también permita ser ampliada para no perder vigencia en el futuro.

Además, la herramienta a construir debe permitir trabajar de forma rápida y sencilla al usuario para que pueda definir el flujo de ETL sin complicaciones.

Finalmente, es importante que la herramienta trabaje de forma eficiente para asegurar que los recursos del servidor sean aprovechados correctamente.

1.3. Objetivos específicos

En base a lo expuesto en el punto 1.2 es que el equipo de trabajo encontró importante trabajar en base a los siguientes objetivos que dan forma al proyecto y posterior construcción de una solución integral.

a) Construir un componente compacto que permita una carga rápida de la herramienta en el explorador web. La carga inicial del componente no debe tomar más de 2 minutos (estándar comercial de respuesta para este tipo de herramientas), usando una conexión de banda ancha (600kbps). La idea es evitar que el usuario tenga que esperar excesivamente para poder comenzar a trabajar con la herramienta.
b) Conseguir una integración total con los otros módulos de la herramienta de forma que el flujo de trabajo sea continuo y uniforme. El tiempo de adecuación para un usuario promedio, al uso básico de la herramienta, no debe ser mayor a 15 minutos (calculado en base al tiempo que toma ingresar tres veces a todas las opciones del menú).
c) Implementar una solución escalable a futuro que permita agregar nuevos tipos de fuente de datos según sea necesario.

1.4. Solución propuesta: Características de la herramienta

La herramienta de Inteligencia de Negocios propuesta está apoyada no sólo en la idea de desarrollar una aplicación útil y funcional, sino en el resultado de una investigación y posterior análisis. De esta forma se logra definir un producto que se ajuste a la realidad empresarial, y que pueda calzar con las necesidades de innovación de la organización.

Se trata finalmente de una herramienta completa de extracción de datos, que se alimenta de distintos orígenes de datos, a los cuales les aplica funciones de transformación y limpieza para poder cargarlos a una base de datos destino alineado bajo el esquema dimensional de modelamiento, donde se podrá iniciar la explotación estratégica.

La aplicación cumple con un ciclo de trabajo que comienza con la extracción de los datos desde diversas fuentes de datos como archivos de bases de datos de diferentes proveedores, hojas de cálculo, archivos planos, entre otros. En base a éstos se ejecuta un proceso de transformación de los mismos mediante la aplicación de reglas predefinidas o personalizadas. Asimismo, la herramienta agrega control de los datos mediante filtros y estandarizaciones. Finalmente, los datos se cargarán en la fuente de destino elegida, donde se efectuarán las validaciones necesarias. En la Figura 1.1 se muestra gráficamente el flujo de procesos que ejecuta la herramienta.

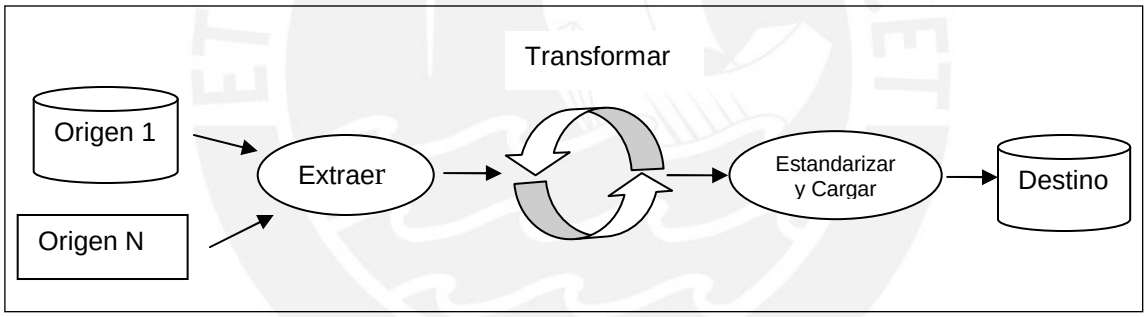


Figura 1.1 : Flujo de procesos de la herramienta

La aplicación es versátil en el sentido que no está sujeta a un estándar específico sino que da libertad a la empresa de poder aplicar sus propios análisis y estructura de orígenes de datos. Esta característica permite un manejo más independiente y personalizado de las funcionalidades. El modelo de ejecución y trabajo dentro de la aplicación está apoyado en criterios de simplicidad y facilidad de manipulación.

La herramienta hace uso de una arquitectura escalable tanto a nivel de *hardware* como en la inclusión de nuevos componentes que puedan formar parte de la solución. Esto representa una cualidad de alto potencial, pues deja la posibilidad y oportunidad de ser éste un proyecto abierto a la inclusión de nuevas formas tecnológicas de manejo de datos.

La herramienta de Extracción de Inteligencia de Negocios permite la creación de

proyectos de trabajo o *Jobs*, los cuales pueden guardarse y recuperarse durante el modelamiento de los flujos de extracción de datos. La ejecución de estos *Jobs* puede hacerse cuando uno requiera, en forma manual, o también programarse para su ejecución posterior, mediante la calendarización de la ejecución. Esto permite ejecutar procesos de extracción complejos, en periodos donde el tráfico de las transacciones en las bases de datos de negocio sea bajo.

La herramienta puede alimentarse no sólo de una base datos especializada como por ejemplo la de Oracle 8, sino que puede recibir también datos de archivos de texto, XML y archivos de hoja de cálculo Microsoft Excel [1].

Los algoritmos que forman parte del motor de filtrado, transformación y estandarización de datos desarrollan procesos ágiles pero consistentes. Todos los datos temporales que puedan generarse durante la ejecución de los flujos de extracción son trabajados de manera interna por el motor, sin necesidad de alterar las bases de datos de origen o destino.

1.5. Productos existentes

Dentro los productos existentes que ofrecen funcionalidades similares encontramos los siguientes:

- Sagent ETL

  Este sistema integrado extrae, transforma, mueve, distribuye y presenta la información clave para la toma de decisiones en la empresa. [SAG 2006]

- MicroStrategy

  Este *software* cubre los requerimientos de *reporting*, análisis y capacidades de envío en una sola plataforma, que permite, entre otras funciones, la gestión de seguridad centralizada, la administración, el desarrollo y la implementación de soluciones de *Business Intelligence* en forma centralizada. [MST 2006]

- Business Objects

  Esta herramienta permite a los usuarios el acceso, análisis y distribución de la información. Business Objects se caracteriza por ser una herramienta fácil de usar, segura, escalable y extensible. Incluye soluciones de consulta, generación de informes y análisis, un portal de BI con funcionalidad completa de *broadcasting* y potentes herramientas de administración. [BUS 2006]

[1] Oracle 8 y Microsoft Excel son productos y marcas registradas de Oracle Corp y Microsoft Inc, respectivamente.

- Cognos

  Cognos ofrece todas las herramientas para Business Intelligence (BI): *reporting*, análisis, *scorecarding*, *dashboards*, administración de eventos de negocio así como la integración de los datos, en una arquitectura sola, probada. Fácil de integrar, desplegar y utilizar, Cognos entrega un ambiente simplificado de BI que facilita la adaptación del usuario, permite toma de decisiones. [COG 2006]

- Sunopsis

  Sunopsis integra en un solo producto las herramientas necesarias para abordar proyectos de Migración de Datos, Limpieza de datos, ETL en *Batch* o en línea, Replicación de Datos, Sincronización de Datos e Integración de Aplicaciones en línea. Sunopsis se caracteriza por un fácil entorno gráfico, flexibilidad y potencia. [SUN 2006]

- DataStage

  DataStage ofrece una solución ETL eficaz y altamente escalable. Presenta tres características principales: provee de amplia conectividad para acceder fácil y rápidamente a cualquier sistema fuente o destino, ofrece herramientas avanzadas de desarrollo y mantenimiento que aceleran la implementación y simplifican la administración, también presenta una plataforma escalable que puede manejar fácilmente los volúmenes masivos de datos. [DSTG 2006]

Características	SAGENT ETL	COGNOS DECISIONSTREAM	BUSINESS OBJECTS INTELLIGENCE PLATAFORM	SUNOPSIS ETL v3	MICROSTRATEGY	DATASTAGE
1. Data Warehouse Escalable.	✓		✓	✓	✓	✓
2. Proceso de transformación en masa no registro por registro.	✓					
3. Reportes Web.	✓	✓				✓
4. Log de mensajes de error o advertencias.		✓	✓		✓	✓
5. Soporta sentencias SQL.		✓	✓			✓
6. Ejecución de scripts.			✓		✓	✓
7. Conexión con: JDBC, ODBC.			✓	✓		✓
8. Seguridad.	✓	✓	✓	✓	✓	✓
9. Asistente tipo wizard.	✓	✓	✓	✓	✓	✓
10. Administración de proyectos.	✓	✓		✓		
11. Plataforma independiente.				✓	✓	✓

Tabla 1.1: Comparación de productos existentes

2. ANÁLISIS DE LA HERRAMIENTA

Este capítulo se centra en describir los resultados luego de realizar el análisis del entorno y de los usuarios con los cuales interactuará la herramienta a desarrollar.

Además se convierte la lista de requerimientos para la herramienta en una lista de casos de uso que describen la funcionalidad que ofrecerá la herramienta y que satisfacen los requerimientos planteados.

2.1. Diagramas de casos de uso

En esta sección se exponen los principales casos de uso que se deben desarrollar para satisfacer los requerimientos planteados para la herramienta.

Todos los diagramas se han elaborado utilizando el lenguaje de modelamiento UML, como una manera de estandarizar la forma de presentación de la información.

Se han agrupado los casos de uso por afinidad respecto del tipo de funcionalidad que implementarán en la herramienta de manera que la visualización sea más consistente.

La especificación detallada de los casos de uso se encuentra en el documento de Especificación de Requisitos de Software (ERS), anexo C.

2.1.1. Paquete de comunicación

Este paquete agrupa los casos de uso relacionados con la comunicación a las fuentes de datos con las cuales trabajará la herramienta durante la ejecución de las transformaciones. Estas fuentes pueden hacer la vez de origen de datos o destino de datos y deben ser

configuradas apropiadamente antes de poder ser utilizadas por la herramienta. Se muestra el diagrama en la figura 2.1.

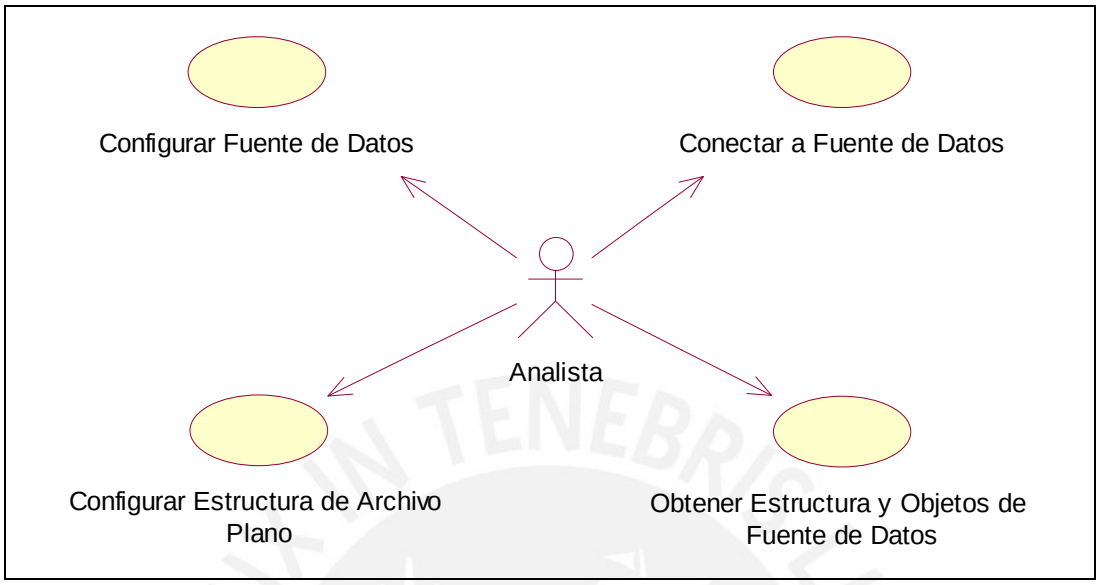


Figura 2.1: Diagrama de casos de uso – Paquete comunicación

Configurar fuente de datos: El propósito de este caso de uso es el de permitir al usuario configurar una fuente de datos. Luego de ser configurada, la fuente de datos podrá ser utilizada para formar el flujo de transformación de datos.

Conectar a fuente de datos: El propósito de este caso de uso es el de permitir al usuario conectarse a una fuente de datos configurada en el sistema.

Obtener estructura y objetos de fuente de datos: El propósito de este caso de uso es el de permitir al usuario obtener las estructuras y objetos de las fuentes de datos para poder utilizarlos en la creación del flujo de transformación.

Configurar estructura de archivo plano: Permite al usuario definir la estructura de campos que conforman un archivo plano y convertirlo a objetos que puedan ser utilizados en la construcción del flujo de transformación.

2.1.2. Paquete de extracción

Este paquete agrupa los casos de uso relacionados a la configuración del flujo de extracción y transformación de datos. Mediante estos casos de uso el usuario define cómo se realizará la extracción de datos, desde cuáles orígenes y hacia qué destinos, y qué transformaciones se harán sobre los datos en el proceso.

También se incluyen los casos de uso relacionados a la creación de los niveles jerárquicos de organización de los flujos de transformación: *jobs* y paquetes; y casos de uso para funcionalidades adicionales que se utilizan durante la etapa de creación del flujo como son la configuración de parámetros, el orden de ejecución de los paquetes dentro de un *job* y definición de funciones definidas por el usuario. Se muestra el diagrama en la figura 2.2.

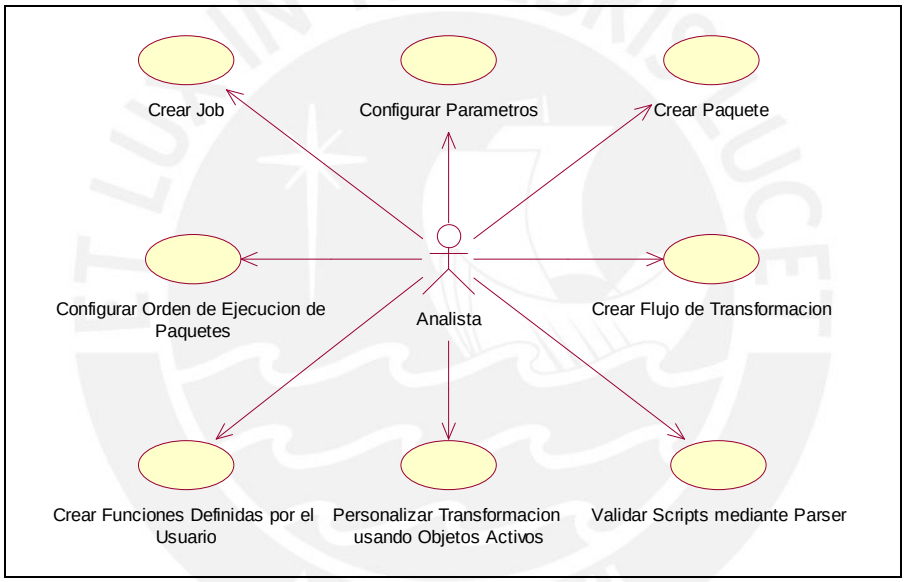


Figura 2.2: Diagrama de casos de uso – Paquete extracción

Crear *job*: El propósito de este caso de uso es el de crear un *job* que permite la administración de paquetes dentro de un proyecto.

Configurar parámetros: El propósito de este caso de uso es el de crear los parámetros que serán usados en la ejecución de los paquetes.

Crear paquete: El propósito de este caso de uso es el de crear un paquete de extracción dentro de un *job* previamente configurado.

Configurar orden de ejecución de paquetes: El propósito de este caso de uso es el de

permitir al usuario definir el orden en que se ejecutarán los paquetes que pertenecen a un *job*.

Crear flujo de transformación: El propósito de este caso de uso es la definición del flujo de transformación de un paquete.

Crear funciones definidas por el usuario: El propósito de este caso de uso es el de permitir al usuario definir sus propias funciones para la transformación y estandarización de datos en la configuración de los objetos activos del flujo de transformación en un paquete. Hay dos tipos principales de funciones que se pueden definir: por sentencia y por reglas.

Personalizar transformación usando objetos activos: El propósito de este caso de uso es el de permitir configurar la forma como se realizará la transformación de datos desde los orígenes a los destinos del flujo de transformación.

Validar *scripts* mediante *parser*: El propósito de este caso de uso es el de verificar que no haya errores de sintaxis en los *scripts* ingresados por el usuario o incluidos en el flujo de transformación.

2.1.3. Paquete de ejecución

Este paquete agrupa los casos de uso relacionados a la ejecución de un *job* en el servidor de aplicaciones de la herramienta. Se muestra el diagrama en la figura 2.3.

La ejecución puede ser iniciada manualmente o automáticamente de acuerdo a una programación configurada por el usuario.

Durante la ejecución se genera un *log* de ejecución el cual registra todas las acciones realizadas durante la ejecución, así como también los errores que pudieran ocurrir. Este *log* puede ser visualizado en tiempo real o revisado luego de la ejecución pues se guarda el histórico de *logs* en el servidor, creando un archivo *log* por cada ejecución realizada.

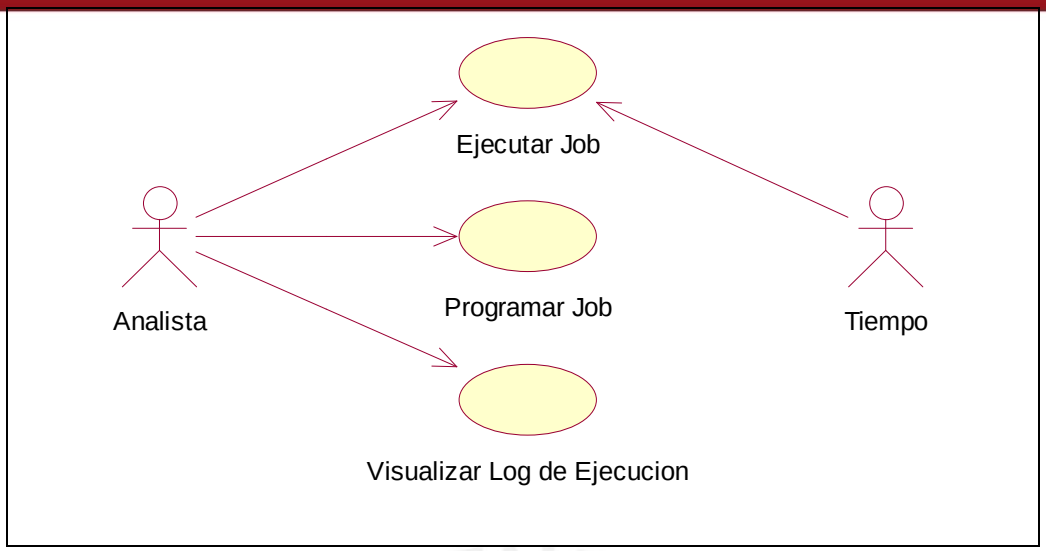


Figura 2.3: Diagrama de casos de uso – Paquete ejecución

Ejecutar *job*: El propósito de este caso de uso es el de ejecutar un job programado en el servidor y generar el *log* de ejecución.

Programar *job*: El propósito de este caso de uso es el de permitir al usuario programar un *job* para su ejecución en el servidor en una fecha y hora definidas, con una periodicidad específica.

Visualizar *log* de ejecución: El propósito de este caso de uso es el de permitir al usuario visualizar los pasos que se han ejecutado en el servidor durante la ejecución de un *job*.

2.2. Características de los usuarios

Se tienen dos usuarios para la aplicación: Analista y Administrador. A continuación se detalla las características y funcionalidades de la herramienta con las cuales interactúa cada uno.

2.2.1. Analista

Este usuario es el actor principal en la ejecución del programa, ya que es el que se encargará de realizar el análisis de su empresa, y según ese análisis armar el flujo de datos que conecte la base de datos transaccional (origen) con el *Data Warehouse* (destino) que pueda cumplir con todos los requisitos planteados en la etapa de análisis.

Por lo dicho anteriormente, no es necesario que este usuario posea conocimientos avanzados sobre informática, puesto que el uso de la aplicación es intuitivo y no requiere grandes habilidades para utilizarla. Lo que sí es muy importante es que este usuario tenga sólidos conocimientos en Inteligencia de Negocios. Tiene que ser capaz de diferenciar lo que se tiene en el origen, y lo que desea obtener en el destino, de forma que pueda armar el flujo que conecte a ambos de la manera más simple posible. Dicho flujo deberá ser construido por el analista usando todos sus conocimientos del negocio.

También será necesario que el analista conozca todas las posibilidades que brinda la aplicación para realizar el flujo, lo que significará obtener los datos en el destino más rápidamente y obtener los respectivos reportes en el momento preciso en que se necesiten, sin retrasos. Del analista depende entonces la forma y la precisión de los datos que se obtengan en el destino.

Como la aplicación será usada por varios analistas, que pueden estar en el mismo proyecto (Ver anexo C – Especificación de Requisitos de Software) o en diferentes, es necesario que éstos dividan el trabajo de forma que se evite hacer una misma tarea dos veces, la aplicación está hecha de tal manera que permite a los usuarios crear diversos objetos o funciones reutilizables en sus proyectos, por lo que esta funcionalidad debe ser explotada al máximo por los analistas durante el desarrollo de los mismos. Además, como la aplicación se ha construido para que varias personas puedan trabajar simultáneamente, se encarga de proteger a los usuarios de pérdida del trabajo realizado en los flujos alertándolos cuando el flujo está siendo trabajado por otro usuario. Esto se logra abriendo también el paquete o flujo en modo de lectura.

2.2.2. Administrador

Este usuario vendría a ser el actor secundario en la ejecución del programa, aunque cuente con mayores privilegios que el analista. Se encargará de administrar los perfiles del Sistema, y otorgar a cada perfil los permisos que convenga según la posición en el proyecto. De esta forma se asegura un mejor orden durante el desarrollo del proyecto.

Además este usuario debe encargarse de administrar el servidor Web y asegurar su funcionamiento las 24 horas del día. La ejecución ininterrumpida del servidor durante el día completo es necesaria puesto que se tienen programaciones hechas por los usuarios para ejecutar sus flujos en cualquier hora del día, en cualquier día de la semana. Si por algún motivo una programación hecha por algún usuario no se pudo ejecutar porque el servidor no estaba disponible, el administrador tiene la responsabilidad de reprogramar el *job* para ejecutarse en otro momento.

Como la metadata de la aplicación no se guarda en base de datos, sino en archivos planos, esto facilita la tarea del administrador de cuidar dicha información, puesto que los archivos planos implican menos complejidad en comparación con las bases de datos, y su uso es apropiado teniendo en cuenta la cantidad de metadata que se va a manejar [BOU 2005].

3. DISEÑO DE LA HERRAMIENTA

Este capítulo se concentra en presentar el diseño de la herramienta. Se incluyen los diagramas de las diferentes partes que componen esta etapa, elaborados utilizando el lenguaje de modelamiento UML.

La etapa de diseño del proceso de desarrollo del *software* es la que se realiza luego de completar el análisis, y traslada los resultados obtenidos en la etapa previa a información más específicamente ligada a la siguiente etapa de construcción. Para este fin, se elabora un diagrama de clases de diseño, que representa las clases que serán programadas en la etapa de construcción.

Una clase de análisis puede dar como resultado en el diseño la creación de muchas clases de diseño, pues se debe considerar que para manejar la clase base (la que contiene los datos del análisis), se necesita en la mayoría de ocasiones contar con una clase gestora; y para permitir el manejo de los gestores desde la interfaz de usuario de forma ordenada, siempre es buena idea contar con clases controladoras. Además se debe incluir las clases que implementan la lógica central de procesamiento de datos de la herramienta. Entonces, se puede tener una idea de cuánto más puede crecer el diagrama de clases cuando se pasa del análisis al diseño.

Otro punto importante a resaltar en la etapa de diseño es que las clases reciben el nombre con el cual serán implementadas en la programación pues, como se indicó líneas arriba, en esta etapa los datos presentados deben estar más ligados a la construcción.

Como se trata de un proyecto de mediana envergadura, con varios módulos que interactúan entre sí y un equipo de desarrollo numeroso (13 tesistas), se hace imperativo contar con un estándar común para ordenar y nombrar las clases de programación. En este estándar

se definieron las siguientes reglas para los nombres de clases:

a) Las clases llevarán un prefijo de dos letras de acuerdo al módulo al cual pertenecen: BA (análisis), BE (extracción), BX (explotación). Si son clases comunes utilizadas en más de un módulo no se les colocará prefijo.

b) Para las clases gestoras se utilizará la palabra 'Gestor' seguido del tipo de objeto del cual es gestor. Ejemplo: BEGestorTabla.

c) Para las clases controladoras se utilizará la palabra 'Control' seguido del tipo de objeto del cual controla funciones. Ejemplo: BEControlTransformacion.

Además se estableció una estructura de organización de las clases en paquetes, para mantener agrupadas las clases de acuerdo a su funcionalidad. Se muestra en la figura 3.1 un esquema del árbol de paquetes a utilizar para la construcción de la herramienta. Ver anexo K – Estructura de paquetes.

Figura 3.1: Estructura de paquetes del proyecto

3.1. Diagrama de clases

Los diagramas de clases permiten al equipo de desarrollo visualizar mejor las relaciones entre los elementos que serán implementados en la etapa de construcción, y sirven como referencia al momento de implementar una clase determinada para tener en cuenta los atributos que se espera que tenga y los métodos que debe implementar para cumplir con las funcionalidades para la cual fue creada.

Dado que se cuenta con una gran cantidad de clases en el proyecto, éstas se han dividido en varios diagramas que las agrupan por afinidad o funcionalidad, de manera que sea más fácil de comprender las relaciones entre clases y evitar sobrecargar el diagrama con demasiados elementos.

Los sub-diagramas de clases que se muestran a continuación son los siguientes: diagrama de clases base, diagrama de clases de conexión a base de datos, diagrama de clases de persistencia, diagrama de clases de dibujo, diagrama de clases de persistencia de dibujo, diagrama de clases del *applet*, diagrama de clases de ejecución, diagrama de clases de manejo de archivos intermedios.

Es importante mencionar que estos diagramas de clases son el resultado de una serie de revisiones y cambios hechos a lo largo de todo el proyecto, durante el cual el equipo de desarrollo fue adquiriendo mayor experiencia y dominio del problema.

Como punto de comparación se muestran en las figuras 3.2 y 3.3 los diagramas de clases que se diseñaron para la versión preliminar construida durante el curso de Desarrollo de Programas 1, cuando el producto estaba en sus etapas iniciales.

Como se puede apreciar en ese diagrama, las clases y sus relaciones son mucho más simples, pero a la vez más rígidas. Este esquema de clases no aprovecha a fondo las bondades de un esquema con clases derivadas o herencias como es el caso de la versión final del producto.

El equipo de desarrollo optó por reorganizar la forma como estaban ordenadas las clases y crear dos categorías principales o clases genéricas: Objetos activos y objetos pasivos.

Los objetos activos corresponden a los que se encargarían de almacenar los datos de configuración de los procesos de transformación de datos que serían ejecutados en el servidor, representados por la clase BEObjetoActivo. De esta clase es que se derivan las clases para cada uno de los subcomponentes que conforman la parte de ejecución del flujo de transformación: BEProceso, BEFiltro, BETransformacion, BEEstandarizacion, BEScript y BELookup.

Los objetos pasivos corresponden a los que representan una fuente o destino de datos y con los cuales interactuará la herramienta, son representados por la clase BEObjetoPasivo. De

esta clase es que se derivan las clases BETabla y BEArchivo, que representan una tabla de una base de datos y un archivo plano, respectivamente. De esta forma, se puede extender en el futuro los tipos de objetos pasivos de los cuales obtener datos con sólo crear una nueva clase derivada de BEObjetoPasivo.

Otro cambio importante en el diseño de la herramienta se encuentra en la forma de manejar la configuración del proceso de transformación de datos.

En la versión preliminar, se había generado un conjunto de clases para contener los datos de configuración de los subcomponentes de filtrado, transformación y estandarización de datos. Estas clases proveían un número limitado de operaciones para cada subcomponente. No había forma de que el usuario pudiera personalizar fácilmente cada subcomponente del proceso de transformación de datos.

Para permitir mayor flexibilidad al usuario de la herramienta, se permitió al usuario ingresar sentencias manualmente a los subcomponentes. Para poder manejar estas sentencias se tuvo que desarrollar un conjunto de clases especializadas que pudieran dividir las sentencias en partes reconocibles por la herramienta. Estas clases son: BESentencia, BEOperacion y BEFuncion, que se muestran más adelante en el diagrama de clases de ejecución, figura 3.11.

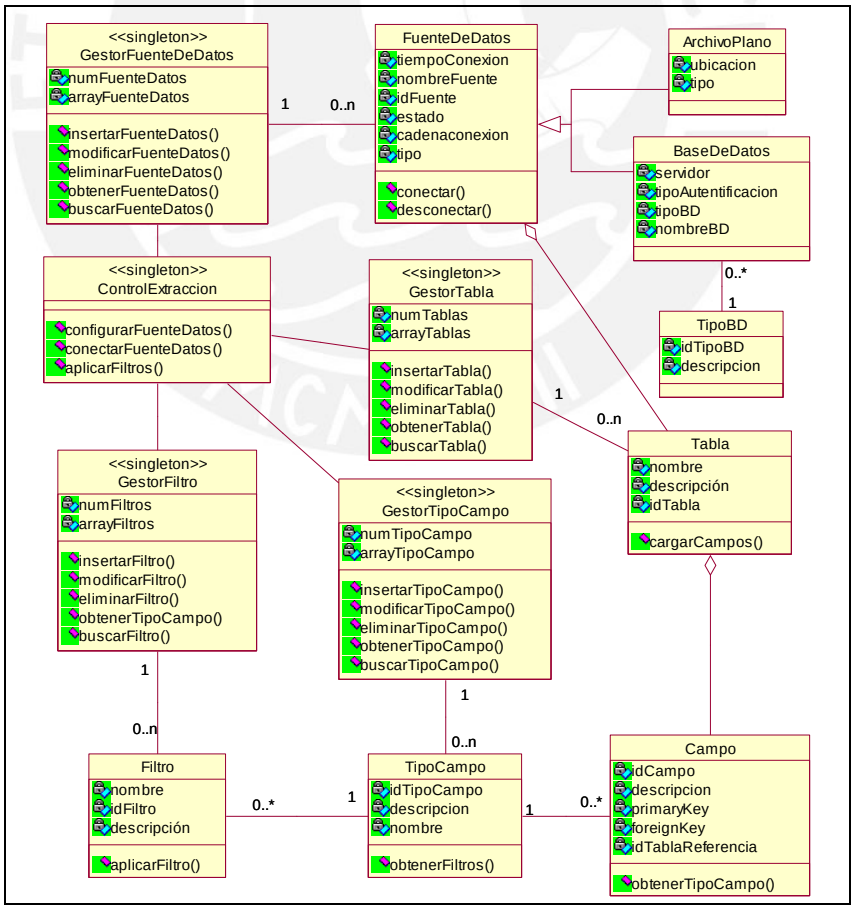


Figura 3.2: Diagrama de clases inicial (parte 1)

Siguiendo con las mejoras entre las etapas de evolución del proyecto, se reformuló la forma como se manejaban las clases gestoras. En la versión preliminar se puede apreciar que las clases gestoras están marcadas como clases *singleton*. Sin embargo, este patrón de diseño resultó inadecuado luego de que se analizó nuevamente el tema. Una clase *singleton* tiene como característica que sólo existe un objeto instanciado de esta clase en el sistema, es decir que todas las demás instancias son referencias al mismo objeto. Las clases gestoras deben almacenar una lista de objetos que pertenecen a una clase específica, por ejemplo el gestor de tablas contiene la lista de todos los objetos del tipo Tabla en el sistema y permiten ubicar un objeto dentro de su lista mediante métodos de búsqueda apropiados.

El problema que trae esta forma de trabajo es que a largo plazo el número de objetos registrados en el gestor crece considerablemente haciendo que las búsquedas se hagan mucho más lentas, degradando la performance general de la herramienta.

La solución dada es que cada objeto instanciado tenga su propio juego de gestores para los objetos que le pertenecen. Por ejemplo, un paquete puede tener muchas fuentes de datos y también muchos componentes. Entonces, el objeto BEPaquete tiene una referencia a su propio BEGestorFuenteDatos para manejar sus objetos fuente de datos y otra a su propio BEGestorComponente para manejar sus componentes. La ventaja sobre la forma de trabajo descrita es que cada gestor sólo contiene la lista de objetos relacionados al objeto principal. De esta forma, el tiempo de respuesta no es afectado por el crecimiento del volumen de datos en la herramienta.

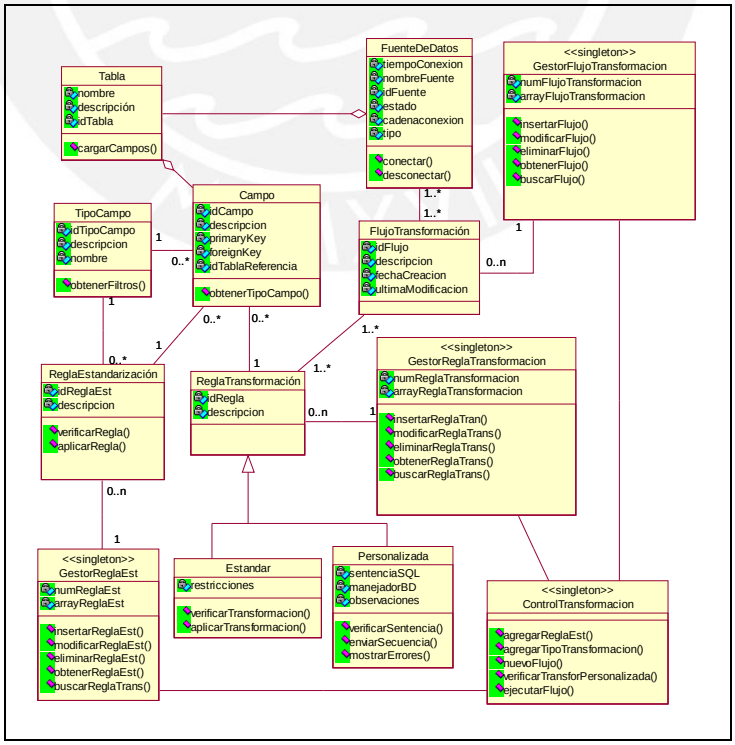


Figura 3.3: Diagrama de clases inicial (parte 2)

En el aspecto de la persistencia de datos, también se hizo un cambio en el manejo de los archivos XML. Se pasó de usar JDOM [JDO 2006] [2] a usar Castor [CAS 2006] [3] debido a que este último permite una transformación más transparente de objetos a archivos XML y viceversa. De esta forma se evitó tener que implementar la escritura y lectura manual de los *tags* del archivo XML para luego instanciar y fijar valores en los objetos adecuados, permitiendo invertir el tiempo ahorrado en otros aspectos más importantes del proyecto.

Para poder trabajar con el Castor se desarrolló un conjunto de *beans* de datos los cuales guardan los valores de los atributos de los objetos que representan. Cada clase *bean* cuenta con dos métodos principales: el método generarBean, que le permite obtener los datos del objeto para cargarlos a sus propios atributos; y el método generarObjeto, que a partir de los datos del *bean* crea la instancia del objeto para ser usada en la herramienta.

Finalmente, otro importante cambio en el diseño de la herramienta corresponde al paso de la arquitectura cliente-servidor de varias capas comunicadas vía RMI (*Remote Method Invocation*) [RMI 2006], a la plataforma web utilizando *applets* y *servlets*. Este cambio supuso un replanteo de las clases controladoras principales pues debían ser accedidas desde un *servlet* y no directamente mediante RMI. Del mismo modo, el componente cliente se tuvo que modificar para canalizar todos los mensajes con el servidor a través de una clase especializada: AppletComunicacion. Más detalles en la sección 3.1.6, Clases del Applet.

3.1.1. Clases base

Las clases base son las clases que contienen los datos que maneja la herramienta para las diferentes funcionalidades que ofrece. En su mayoría contienen datos que deben ser persistentes, es decir, deben ser almacenados en un medio no volátil para poder ser recuperados en el futuro y seguir trabajando con ellos en la aplicación. Se muestra el diagrama en la figura 3.4.

Cabe resaltar que todas las clases base heredan de la clase Objeto, el cual contiene los atributos básicos que todas las clases deben tener. Estos son: id (identificador único), nombre, descripción, fecha de creación y fecha de modificación. Del mismo modo, se ha aplicado herencia en varias clases del módulo de extracción para facilitar el manejo de tipos de datos similares en los diferentes controles de la herramienta. Por ejemplo podemos destacar las clases BETabla y BEArchivo, las cuales heredan de la clase BEObjetoPasivo.

La clase BETabla representa una tabla de base de datos, mientras que la clase BEArchivo representa un archivo plano, ambas clases son del tipo BEObjetoPasivo pues pueden ser origen o destino de datos para el procesamiento del flujo de transformación. De

[2] El proyecto JDOM provee una solución para manipular directamente la estructura de un archivo XML desde código Java.
[3] El proyecto Castor es una plataforma de enlace de datos (d*atabinding*) que permite convertir objetos Java en archivos XML y viceversa de forma simple y transparente, llamando a métodos de una clase especial, evitando tener que manipular manualmente la estructura de los archivos XML.

esta manera se simplifica la ejecución de un flujo en el control de transformación, pues se considera de igual manera el tener una tabla o un archivo como origen de lectura de datos.

En este diagrama de clases se incluyen también las clases gestoras que se encargan de administrar el manejo de los objetos de clases base. Estas clases gestoras cuentan con los métodos apropiados para permitir buscar un objeto dentro del conjunto de objetos que contiene, así como para agregar, modificar o eliminar un objeto específico.

En algunos casos especiales se tiene clases base que son a la vez clases gestoras pues sólo administran un tipo de objeto a diferencia de las otras clases que requieren de una clase gestora separada. Este es el caso de la clase BEJob, la cual solo administra objetos de tipo BEPaquete, por lo que se hace innecesario crear una clase gestora especializada en administrar objetos del tipo BEPaquete. Si se ve en cambio la clase BEPaquete, se notará que cuenta con tres clases gestoras asociadas para poder administrar los objetos que pertenecen al paquete. Estas son BEGestorFuenteDatos, BEGestorComponente y BEGestorFuncion.

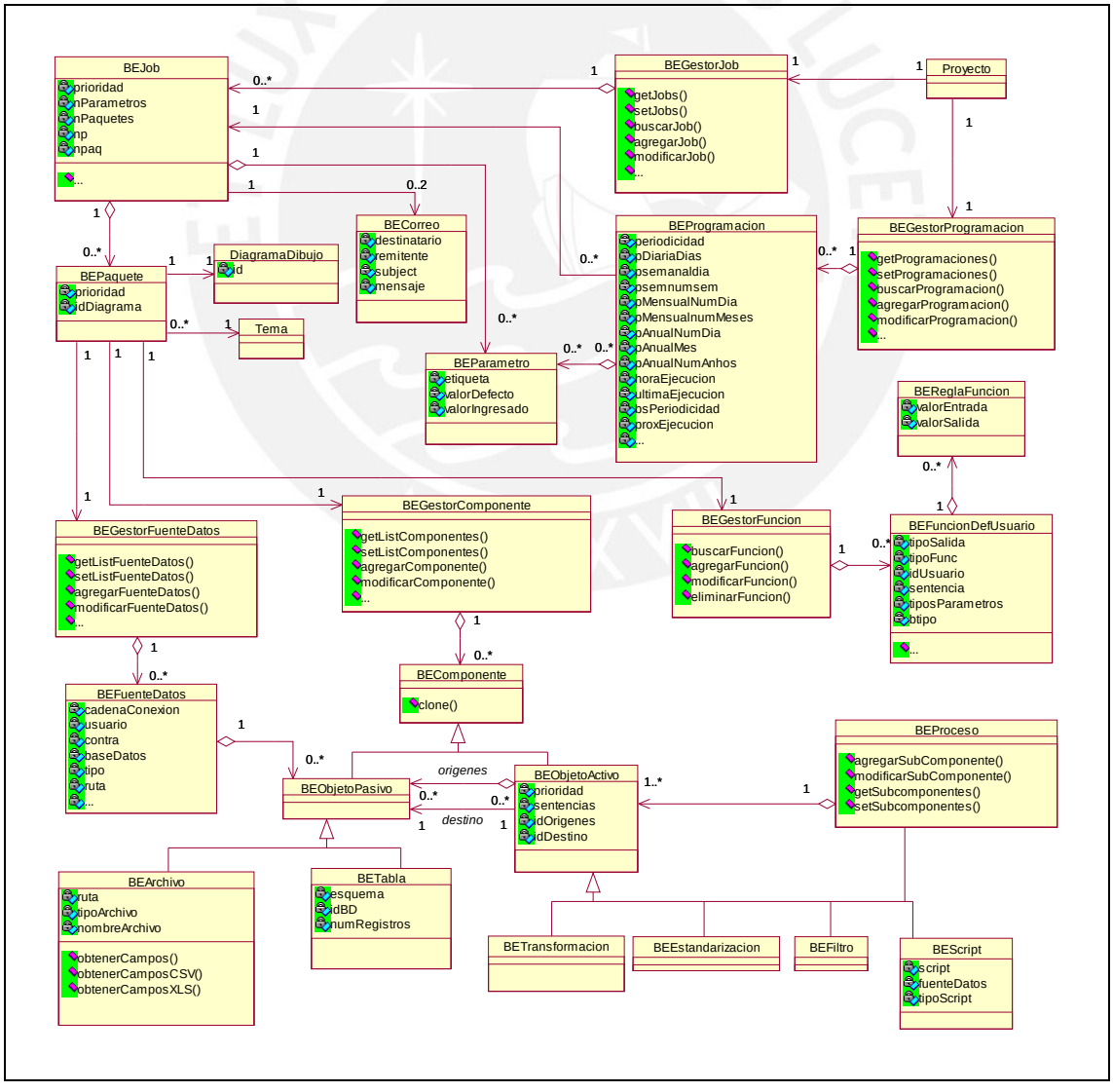


Figura 3.4: Diagrama de clases base

3.1.2. Clases de conexión a base de datos

Las clases de conexión a base de datos son las encargadas de gestionar la conexión a una base de datos, obtener la estructura de tablas y objetos y realizar la interacción entre la herramienta y la base de datos ejecutando sentencias SQL en el motor de base de datos.

Además, permiten almacenar la información para permitir la conexión a una base de datos mediante la clase BEFuenteDatos, así como almacenar información de la estructura de la base de datos mediante el uso de las clases BETabla, BECampo, BEFormato, BETipoDato.

Esta organización de clases para describir la estructura de la base de datos es fundamental para la correcta ejecución de los flujos de transformación pues permiten a la herramienta determinar que campos se leerán o escribirán en una base de datos, así como conocer el tipo de dato de cada campo y de esta forma utilizar el método de inserción o lectura más apropiado y la función de transformación que permita manejar correctamente este tipo de dato. Se muestra el diagrama en la figura 3.5.

Cabe resaltar en esta sección que el control de fuente de datos está configurado para utilizar *drivers* externos que permiten la conexión a bases de datos. La idea de usar *drivers* externos es que se pueda extender en un futuro el número de bases de datos a las que se pueda conectar la herramienta, sólo sería necesario agregar el *driver* apropiado al proyecto y configurarlo en el BEControlFuenteDatos.

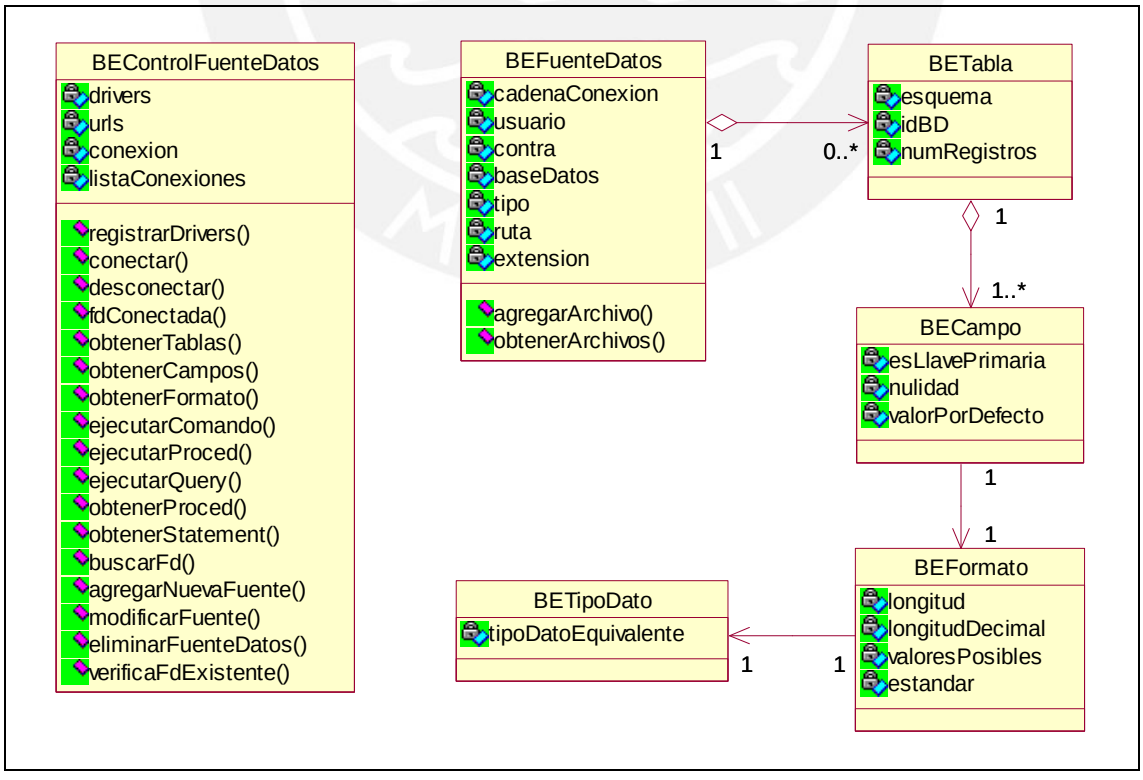


Figura 3.5: Diagrama de clases de conexión a base de datos

3.1.3. Clases de persistencia

Las clases de persistencia son las encargadas de convertir los datos de las clases base en archivos XML que serán almacenados en el servidor para poder recuperar la información en otro momento. Hay una clase de persistencia por cada clase base que sea necesario grabar o leer de archivos XML. Se muestra el diagrama en la figura 3.6.

La clase principal que se encarga de convertir los datos de las clases base en clases de persistencia es BEControlXML. Esta clase se comunica con XmlMaker, el cual convierte los datos de los objetos java en archivos XML.

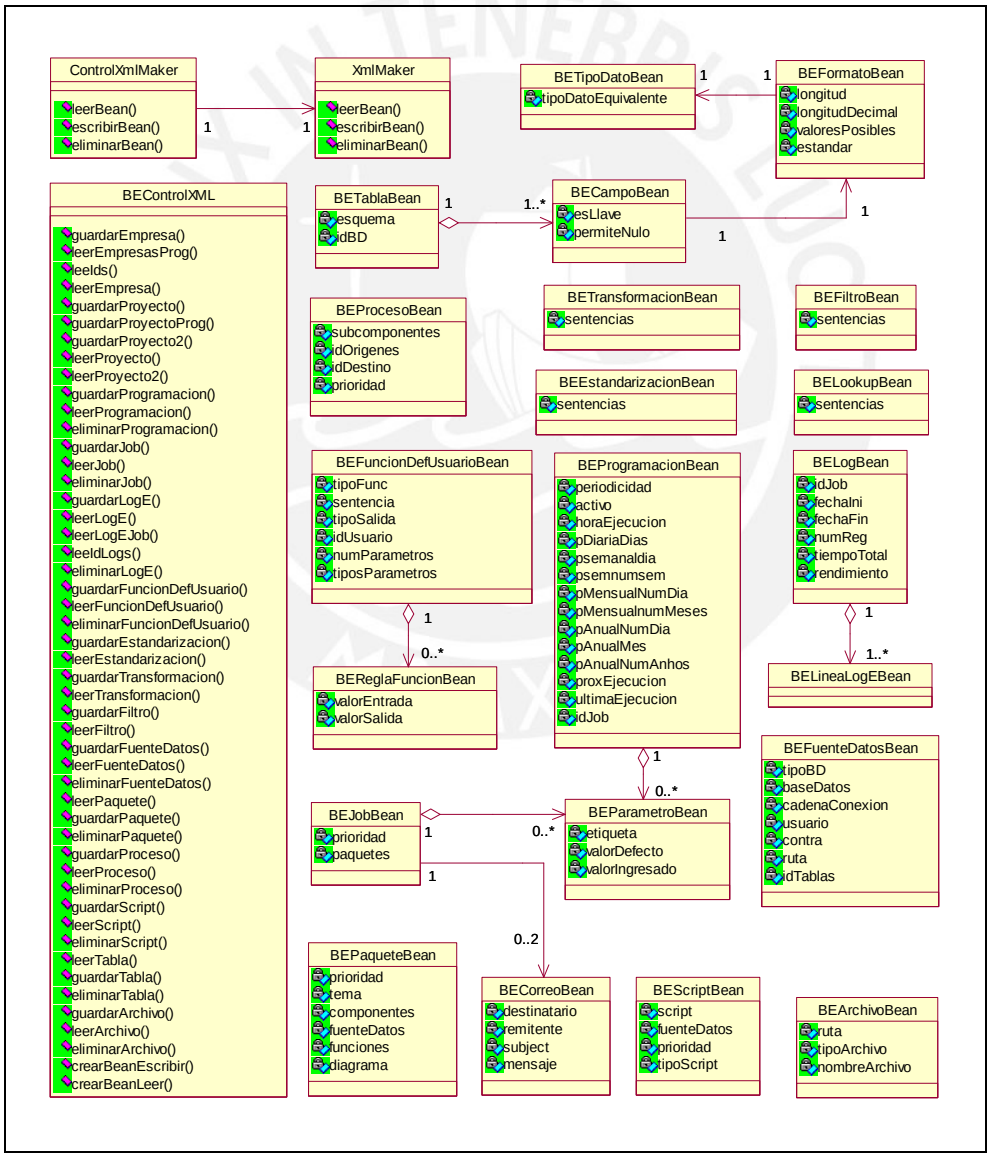


Figura 3.6: Diagrama de clases de persistencia

Los archivos XML que se almacenen en el servidor estarán organizados en carpetas de acuerdo al tipo de objeto al que pertenezca el archivo XML.

En un archivo XML pueden almacenarse más de una clase a la vez, como es el caso de la clase BEProcesoBean. En el archivo XML que corresponde a esta clase, se incluye además los datos de los subcomponentes que lo conforman como son el filtro, transformación, estandarización y *lookup*. Otras clases solamente guardan la referencia a los objetos relacionados dado que el objeto relacionado es referenciado por más de un objeto y no sería adecuado guardar una copia del mismo objeto en cada archivo XML del objeto que le referencia. Esto último porque podrían ocurrir problemas de sincronización, al guardar en cada objeto una versión diferente del objeto relacionado, provocando que los datos no sean consistentes.

A continuación se muestra la estructura de carpetas en las que se organizarán los archivos XML.

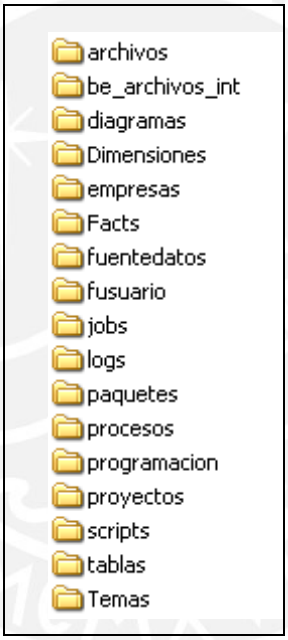


Figura 3.7: Estructura de carpetas de archivos XML

La descripción de cada carpeta y la estructura interna de los archivos XML se especifica en el Documento de XML, anexo G.

3.1.4. Clases de dibujo

Las clases de dibujo son las que permiten mostrar en el diagrama del *applet* los diferentes componentes que conforman el flujo de transformación de un paquete. Implementan métodos para responder a eventos del *mouse* tales como arrastrar, doble clic y clic derecho en el objeto. Se muestra el diagrama en la figura 3.8.

Todos los objetos que conforman el diagrama de dibujo y que son mostrados en pantalla

heredan de la clase JComponent, aprovechando que tiene muchas funcionalidades relacionadas con el arrastre de objetos y manejo de eventos implementados las cuales permitieron construir sobre éstos, métodos más específicos para la herramienta.

Aquí también se hace uso extensivo de la herencia de clases para aprovechar el polimorfismo en los métodos que se implementaron de la clase JComponent. En especial fue de gran utilidad para los casos de manejo de eventos del *mouse*.

Es de especial importancia notar que los objetos de dibujo de la clase ObjetoDib tienen como atributo el identificador del objeto lógico que está relacionado al gráfico. De esta manera es posible obtener los datos de un objeto BETabla al hacer doble clic sobre un objeto BETablaDib, pues existe este vínculo entre el objeto lógico y el gráfico.

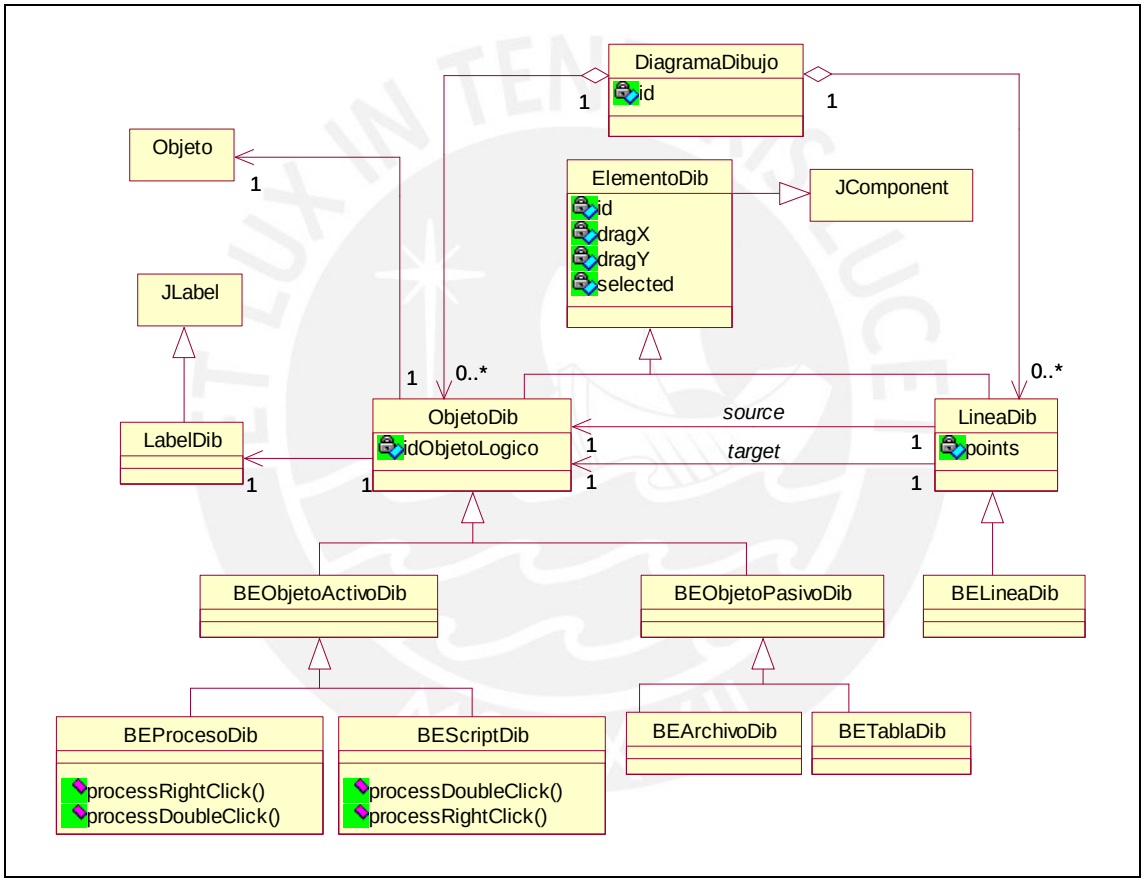


Figura 3.8: Diagrama de clases de dibujo

3.1.5. Clases de persistencia de dibujo

Las clases de persistencia de dibujo, de manera similar a las clases de persistencia mencionadas en la sección 3.1.3, se encargan de convertir los objetos de dibujo en objetos que puedan ser almacenados en archivos XML. Se muestra el diagrama en la figura 3.9.

La carpeta 'diagrama' es la que contiene los archivos XML que tienen la información de

los diagramas que corresponden a los flujos de transformación.

Cada archivo XML representa un diagrama de dibujo y contiene todos los elementos de dibujo del diagrama, objetos gráficos y relaciones entre objetos, guardando la información de posición, tamaño y una referencia al objeto lógico relacionado al objeto gráfico.

La clase principal que se encarga de convertir y administrar los objetos de dibujo que se convertirán a XML es BEControlDibXML. Esta clase es instanciada del lado del cliente, en el *applet* de la herramienta y se comunica con un *servlet* del lado del servidor enviándole los beans de persistencia de objetos de dibujo para que el servidor se encargue de guardarlos en archivos XML.

Es importante que el flujo de datos se realice de esa manera pues algunos de los objetos de dibujo que forman el diagrama no son serializables, lo que significa que no pueden ser transformados para ser enviados a través de la red. Por esta razón es imprescindible que se transforme los objetos de dibujo a beans de persistencia en el lado del cliente y que se envíen los beans al servidor.

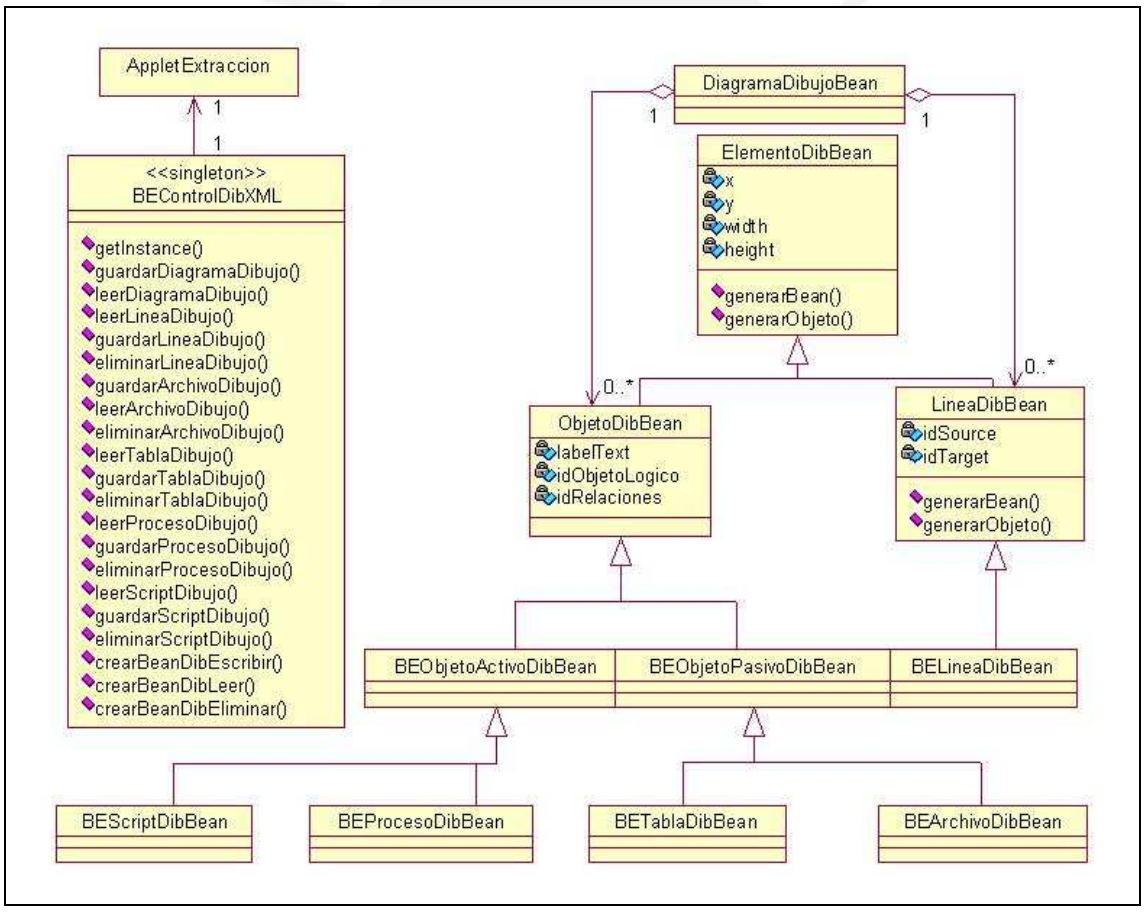


Figura 3.9: Diagrama de clases de persistencia de dibujo

3.1.6. Clases del *applet*

Las clases del *applet* son las que conforman la interfaz gráfica de usuario. Lo conforman la clase AppletExtracción, que contiene toda la GUI; la clase BEDiagramPanel, la cual se encarga del manejo de los objetos de dibujo; la clase BEPanelArbol, la cual se encarga de la administración del árbol de objetos del proyecto; y la clase ControlAccionUsuario que controla las acciones realizadas en el *applet* para permitir la funcionalidad de deshacer y rehacer acciones. Se muestra el diagrama en la figura 3.10.

La clase AppletExtracción hereda de la clase AppletBase, la cual especifica la forma como se distribuyen los elementos gráficos en la pantalla; y ésta, a su vez, hereda de la clase AppletComunicacion, la cual implementa los métodos necesarios para la comunicación entre el *applet* en el lado cliente y el servidor donde se procesan los datos.

Además, en el AppletExtracción se tienen las referencias a los objetos lógicos sobre los que se está trabajando actualmente. Estos son: la empresa actual, el proyecto actual, el *job* actual y el paquete actual.

Finalmente, la clase del *applet* contiene los métodos llamados por las páginas *popup* que permiten configurar los elementos del flujo, como los filtros o transformaciones; así como administrar los objetos del árbol de proyecto, por ejemplo para agregar una fuente de datos o eliminar *jobs*.

La clase BEDiagramPanel es la encargada de mostrar en pantalla los objetos del diagrama de dibujo, controla el factor de *zoom* con el que se muestra el diagrama y registra las acciones realizadas por el usuario enviándolas al control de acciones.

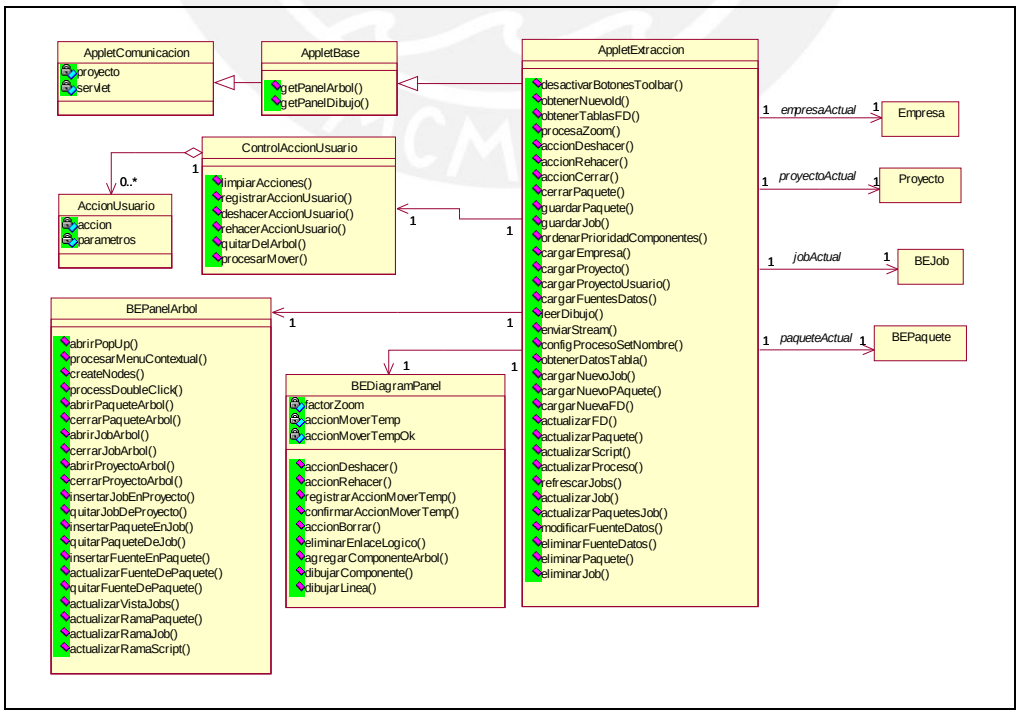


Figura 3.10: Diagrama de clases del applet

3.1.7. Clases de ejecución

Estas son las clases principales que intervienen en los procesos de ejecución de los flujos de transformación definidos por el usuario. Contienen toda la lógica de ejecución y procesamiento de datos e implementan los algoritmos de manejo y transformación de datos fundamentales para el funcionamiento de la herramienta. Se muestra el diagrama en la figura 3.11.

Las clases más importantes que contienen el núcleo de la herramienta son: BEControlProgramacion, BEControlJob, BEControlPaquete, BEControlComponente, BEControlFiltro, BEControlTransformacion, BEControlEstandarizacion y BEControlLookup.

También se muestran en este diagrama las clases que manejan el hilo de ejecución de un *job*, las clases que permiten el uso de funciones definidas por el usuario, el uso de parámetros en la configuración de los *jobs*, el registro de *logs* de ejecución que permita conocer los detalles de una ejecución y el envío de correos electrónicos al finalizar la ejecución de un *job*.

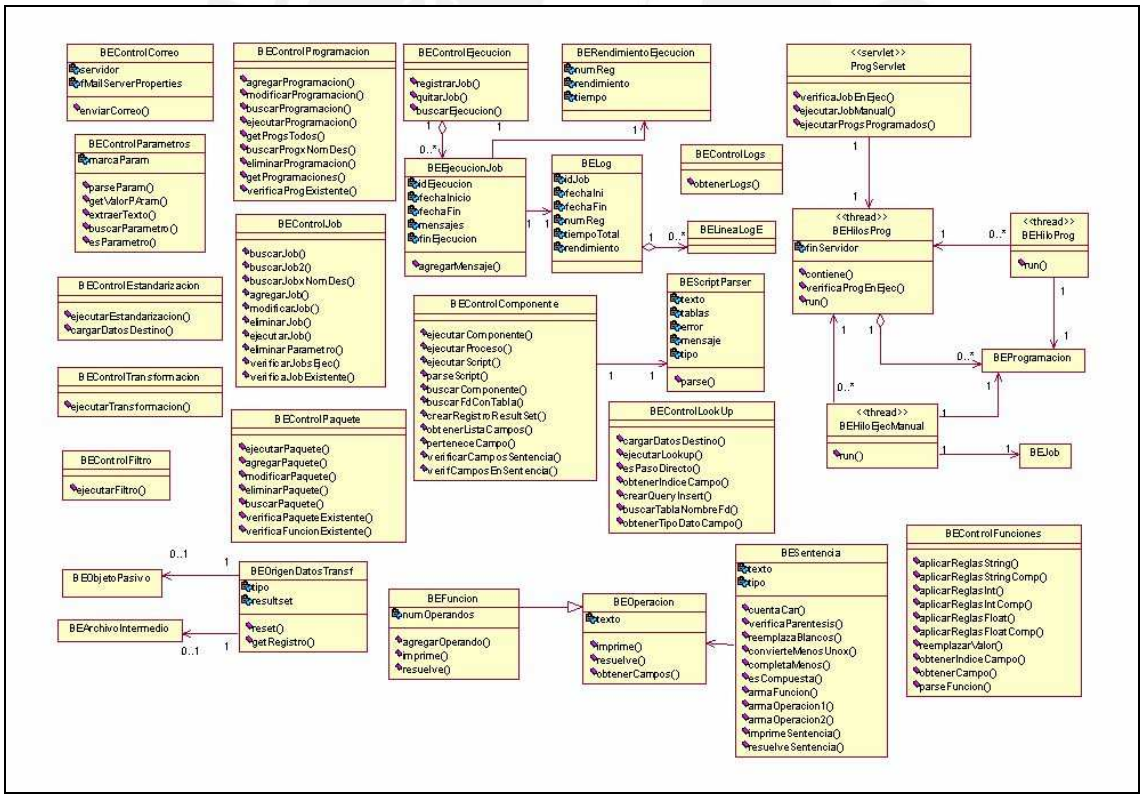


Figura 3.11: Diagrama de clases de ejecución

A continuación se describe una breve explicación del flujo de ejecución: Para verificar si hay algún *job* programado para ejecución se cuenta con un *servlet*, ProgServlet, el cual instancia un hilo que compara la fecha programada con la fecha actual periódicamente. Si se

debe ejecutar un *job*, entonces se inicia un nuevo hilo, BEHiloProg, el cual se encarga de llamar a los métodos de ejecución de los controles que pertenecen al núcleo de ejecución, comenzando por BEControlProgramacion en el nivel más alto y llegando hasta BEControlLookup en la parte más interna del flujo de ejecución.

3.1.8. Clases de manejo de archivos intermedios

Estas clases son parte del proceso de ejecución de un flujo de transformación y son necesarias para poder comunicar las diferentes etapas de la transformación. Así, luego de que se ejecuta el filtrado de datos se escriben los resultados en un archivo intermedio, el cual es leído luego por la etapa de transformación de datos y que, a su vez, genera un nuevo archivo intermedio con los resultados del procesamiento. Este archivo luego es leído por la etapa de estandarización de datos y nuevamente se genera un nuevo archivo intermedio con los resultados de esta etapa. Finalmente, el último archivo intermedio es leído en la etapa de carga de datos o en el *lookup* para poder enviar los datos a su destino.

Al finalizar cada etapa, se borra el archivo intermedio que fue leído para minimizar el espacio ocupado en el servidor. Se muestra el diagrama en la figura 3.12.

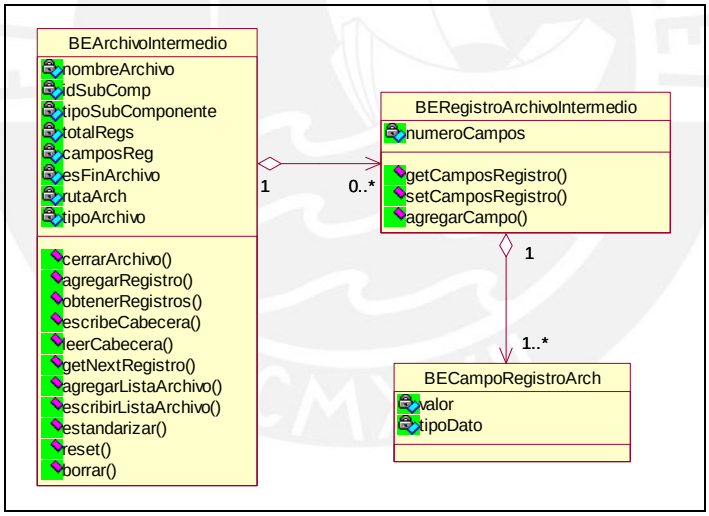


Figura 3.12: Diagrama de clases de manejo de archivos intermedios

3.2. Diagramas de secuencias

Los diagramas de secuencias permiten modelar la comunicación entre clases que se requiere para completar la ejecución de una funcionalidad de la herramienta.

De esta manera se puede visualizar las relaciones y dependencias que existen entre las clases e identificar las clases que intervienen para implementar una funcionalidad.

Son muy útiles cuando se requiere ubicar la fuente de algún problema pues permite ver

claramente en que punto podría encontrarse la falla, observando el flujo de llamadas entre clases.

A continuación se muestran unos diagramas de secuencias para ilustrar como se han elaborado estos artefactos. De la misma manera que en los diagramas anteriormente mostrados, se utiliza el lenguaje de modelamiento UML.

La lista completa de diagramas de secuencias para los casos de uso implementados en la herramienta se encuentra en el anexo H.

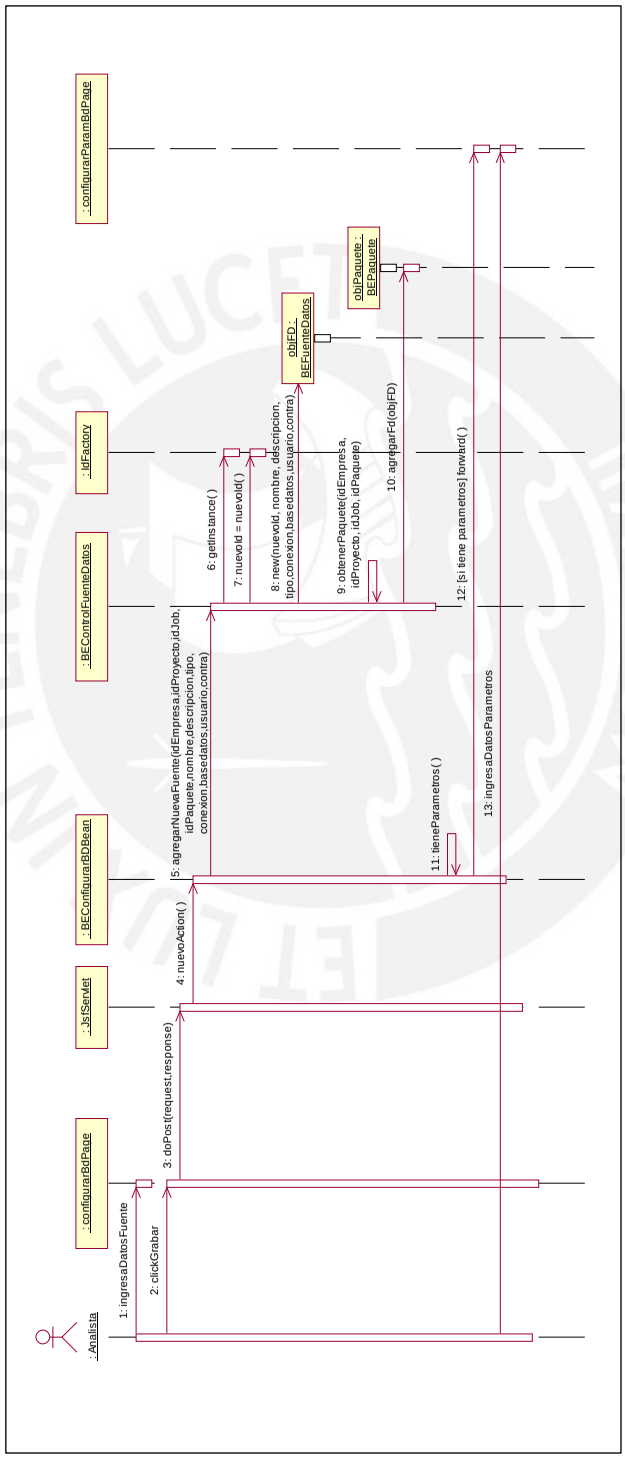


Figura 3.13: Diagrama de secuencias - Configurar fuente de datos

Los diagramas de secuencias pueden volverse muy complejos si la funcionalidad requiere el uso de muchas clases. Como ejemplo está el caso de uso 'Personalizar transformación usando objetos activos', mostrado en la figura 3.14. Aquí intervienen clases para el *applet*, las páginas *popup* de configuración de componente, los *backing beans* de las páginas, los controles de los objetos lógicos que se están configurando, y los objetos a configurar.

Es importante además colocar el suficiente nivel de detalle en las llamadas a los métodos para permitir luego en la fase de construcción la generación rápida del código. Si es necesario se colocan notas para aclarar algunos puntos del diagrama. La idea es que se pueda comunicar con la mayor claridad la forma como interactúan las clases para implementar una funcionalidad.

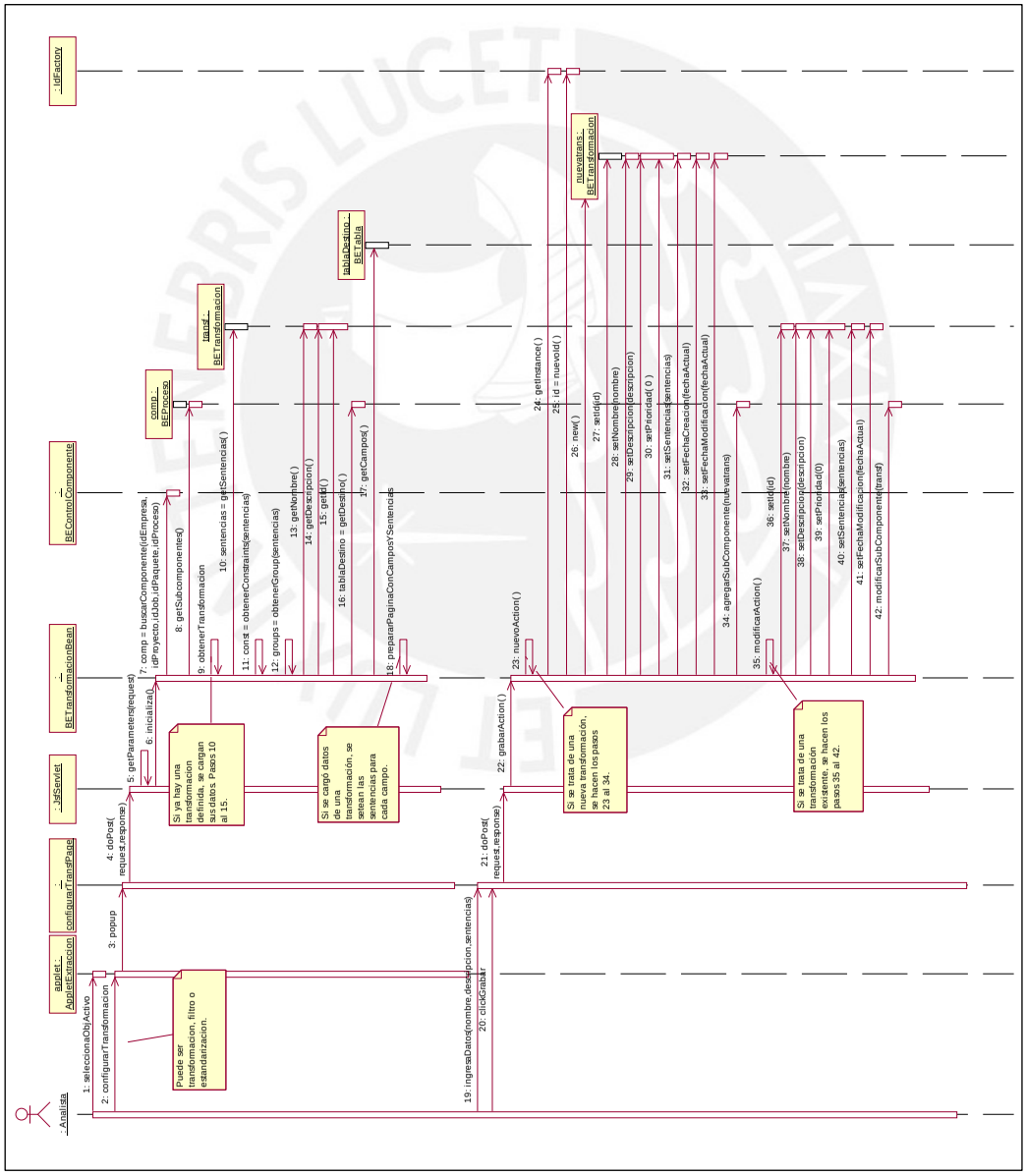


Figura 3.14: Diagrama de secuencias – Personalizar transformación usando objetos activos

3.3. Diagrama de arquitectura

Definir la arquitectura de la herramienta es un punto crítico en el proceso de desarrollo del proyecto pues de ella depende como se implementarán las clases y la forma como interactuarán entre ellas.

Existen diversos modelos de arquitectura que se pueden utilizar para la implementación del software, cada uno con características propias que se adecuan mejor para ser aplicados en una situación determinada.

Al definir el proyecto, se planteó que la herramienta podría ser usada desde cualquier equipo con acceso a Internet. Entonces se decidió utilizar la plataforma web para implementar la herramienta, a través de la cual se proveería el servicio de extracción y transformación de datos. Para cumplir con este requerimiento se podría haber elegido aplicar un modelo tipo cliente-servidor, sin embargo la plataforma web provee mayores ventajas que se mencionan a continuación:

a) Permite acceso a la herramienta desde cualquier lugar del mundo.
b) No se requiere instalar el *software* en el equipo cliente, sólo basta contar con un explorador web con la máquina virtual JAVA y una conexión a Internet.
c) El mantenimiento de la herramienta es más sencillo, sólo se hacen los cambios en el servidor y los clientes descargan la nueva versión del *software* al momento de conectarse. Se evita tener que distribuir la nueva versión del *software* e instalarla en cada estación de trabajo del cliente.

El estilo arquitectónico seleccionado para diseñar la herramienta sobre la plataforma web cuenta con dos componentes principales:

Un cliente ligero, el cual se encarga de toda la presentación visual e interacción con el usuario, así como de realizar algunas validaciones para evitar recargar el servidor. Este cliente se ejecuta en los equipos de los usuarios finales a través del navegador de Internet.

Un servidor central con toda la lógica del negocio y administración de datos. La comunicación con el cliente se realiza a través de mensajes sobre protocolo de hipertexto (*http*) los cuales son transmitidos a través de Internet.

A continuación, se muestran los nodos que conforman la herramienta utilizando el diagrama de despliegue en la figura 3.15.

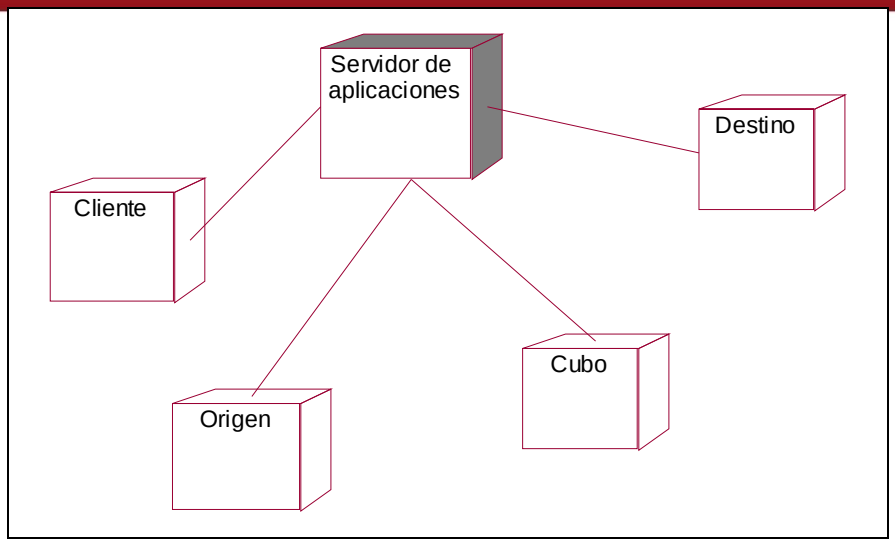


Figura 3.15: Diagrama de despliegue

El Servidor de aplicaciones es la pieza central donde se ubicarán los componentes de la lógica de la herramienta y donde se ejecutarán los procesos configurados por los usuarios para la transformación de datos.

El Cliente es el equipo remoto desde el cual el usuario accede a la herramienta, se conecta con el Servidor de aplicaciones para configurar los flujos de transformación y programar su ejecución.

El Origen y Destino son nodos que corresponden a bases de datos que forman parte del flujo de transformación configurado por el usuario y a los cuales el Servidor de aplicaciones se conectará cuando se ejecute algún proceso de transformación de datos.

Finalmente, el Cubo representa un cubo de datos procesados que le permiten al usuario explotar los datos para obtener información relevante. Este Cubo se muestra en el diagrama de despliegue pues es parte de la herramienta integrada, pero no corresponde al módulo de extracción.

Seguidamente se muestra la organización y relación de los componentes que forman la herramienta AXEbit en el diagrama de componentes en la figura 3.16.

Los principales componentes son los que corresponden a cada módulo de la herramienta: Análisis, Extracción y Explotación. Adicionalmente se cuenta con el componente Control, el cual se encarga de las funcionalidades de seguridad y manejo de sesiones, necesario para poder controlar el acceso de los usuarios a las funcionalidades de los 3 módulos.

Los módulos de Análisis y Extracción utilizan un *applet* como pantalla principal para interacción con el usuario, y cuentan con un área de dibujo para lo cual comparten muchas clases comunes las cuales corresponden al componente de Gráficos.

El componente Común es el que contiene todas las clases comunes a todos los

módulos. Aquí se encuentran las clases que corresponden a un tema de análisis, empresa, proyecto, etc.

Finalmente, los componentes Mondrian y JfreeChart son utilizados por el módulo de Explotación para interactuar con el cubo de datos y presentar gráficos en los reportes.

Una descripción más detallada de la arquitectura y de los componentes y la organización interna de cada uno se puede ver en el Documento de Arquitectura, anexo F.

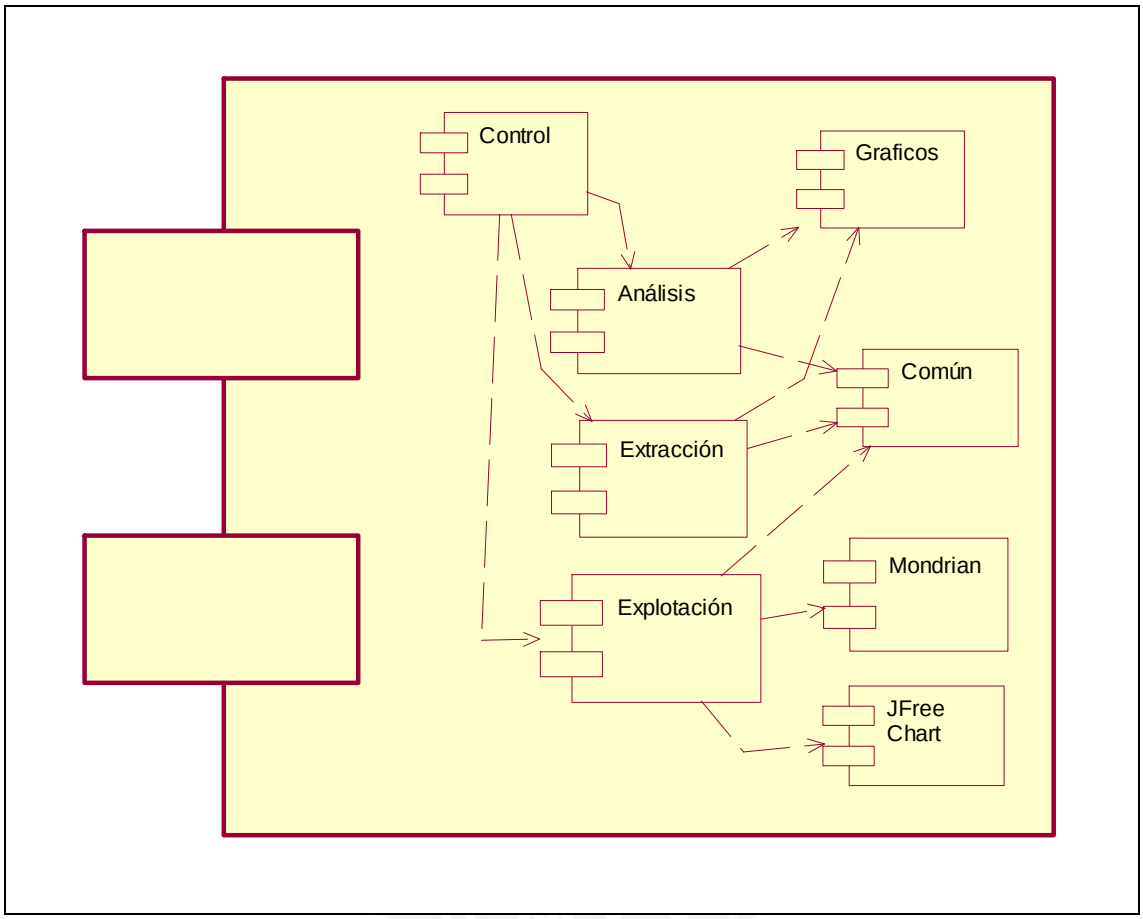


Figura 3.16: Diagrama de componentes

3.4. Diseño de pantallas principales

En esta sección se muestran las pantallas principales que conforman la interfaz de usuario de la herramienta.

Se hace uso conjunto de un *applet* y páginas *popup JSF* (*Java Server Faces*) para implementar las pantallas de usuario para las funcionalidades de la herramienta. Se muestran respectivamente en las figuras 3.17 y 3.18.

En el *applet* se tiene la interfaz principal con la cual interactúa el usuario para el módulo de extracción de la herramienta AXEbit. Las principales partes del *applet* son:

a) Árbol de proyecto: Ubicado al lado izquierdo del *applet*, permite navegar entre

los objetos del proyecto de forma jerárquica, primero la raíz donde se ubican las empresas con las que trabaja el usuario, luego los proyectos que pertenecen a la empresa, en el siguiente nivel los *jobs* y luego los paquetes. Cada elemento tiene un menú contextual que permite configurarlo.

b) Área de dibujo: Corresponde al área más grande del *applet* y es donde el usuario puede formar el flujo de transformación para un paquete. Cuenta con una barra de herramientas en la cual se pueden seleccionar los componentes a agregar al flujo.

c) Menú principal: Ubicado en la parte superior del *applet* permite acceder rápidamente a todas las funcionalidades de la herramienta que estén habilitadas para el elemento activo.

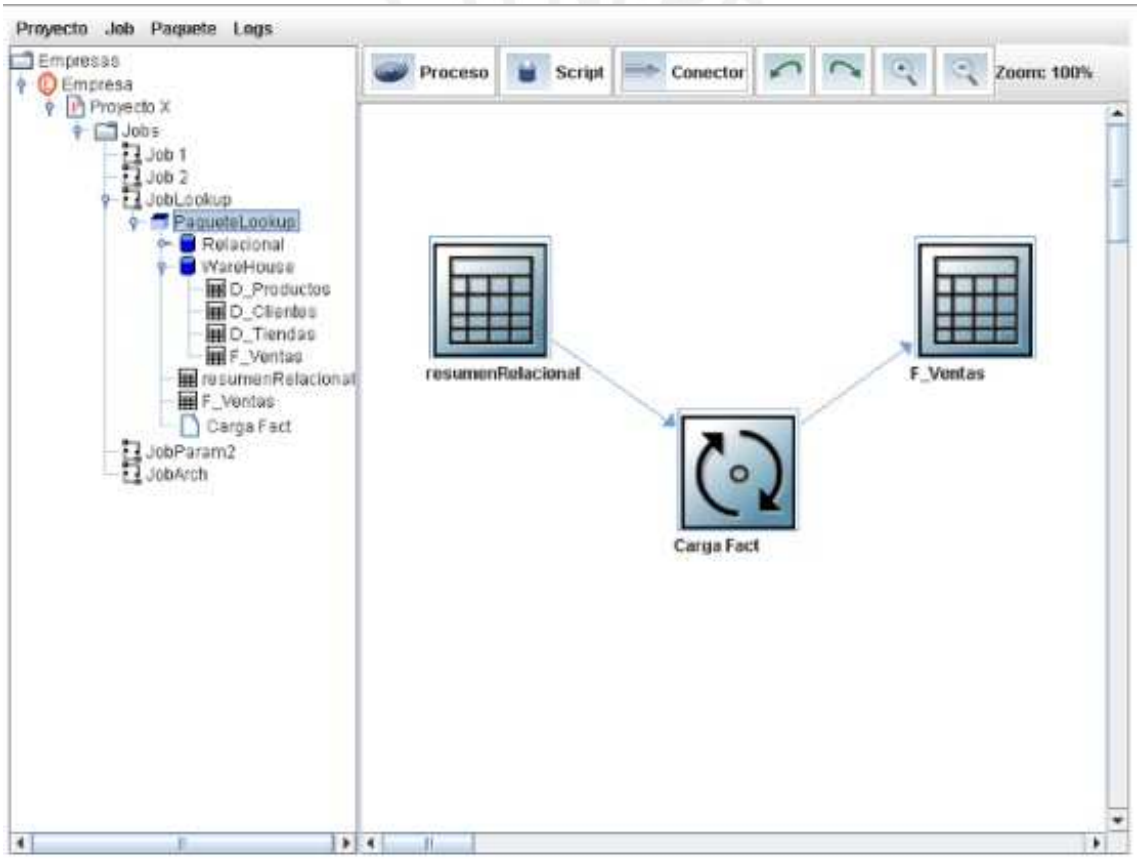


Figura 3.17: Applet del módulo de extracción

Para configurar un componente del flujo de transformación se llama a una ventana *popup* con una página *JSF* apropiada. Del mismo modo se hace para cada funcionalidad del menú, administración de *jobs*, programaciones, muestra de *logs* de ejecución, etc.

Las características principales de las páginas *JSF* que se han implementado para la

herramienta son las siguientes:

a) Se utiliza las hojas de estilo o *cascading style sheets* (CSS) para unificar la definición de estilos de todas las páginas y facilitar de esta forma el mantenimiento de las páginas. Un cambio en la hoja de estilos se verá reflejado en todas las páginas inmediatamente.

b) Se ha seleccionado un color tenue para evitar el agotamiento visual luego de usar la herramienta por tiempo prolongado.

c) Se hace uso de archivos *properties* o *bundle* para obtener el texto que se muestra en las páginas, de esta forma se puede establecer un archivo *properties* por idioma y permitir que se seleccione dinámicamente el archivo de idioma apropiado según la configuración del explorador web que accede a la herramienta. Esto permite que la herramienta soporte múltiples idiomas con un mínimo esfuerzo, pues no hay que modificar las páginas *JSF*, sino solamente definir nuevos archivos *properties* para los idiomas adicionales.

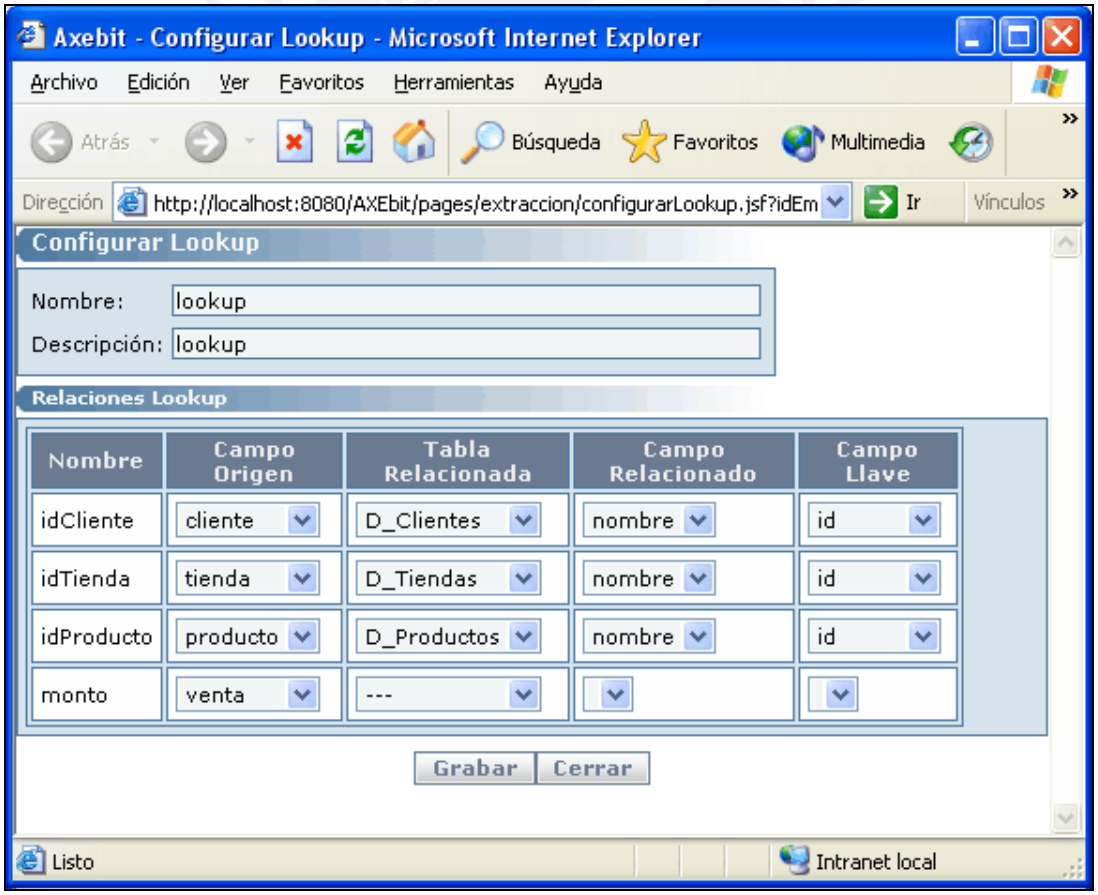


Figura 3.18: Página JSF del módulo de extracción

Una descripción más detallada de las características de las pantallas de la herramienta se puede ver en el Documento de Diseño de Pantallas, anexo I.

3.5. Algoritmos principales

En esta sección se describen los principales algoritmos que se han diseñado para las funcionalidades requeridas de la herramienta. Estos algoritmos permiten entender la lógica de procesamiento de los procesos internos de la herramienta antes de implementarlos en la etapa de construcción.

3.5.1. Conexión a fuente de datos

Este algoritmo permite la conexión a una base de datos especificada por el usuario, así como la obtención de las tablas, campos y datos correspondientes. Los pasos del algoritmo son los siguientes:

Se seleccionó “Fuente de datos”:
- Crear objeto fuente de datos en el proyecto.
- Ingresar datos de configuración de fuente de datos.
- Para la fuente de datos:
    - Crear BEFuenteDatos en base a datos de tabla
    - Generar BEFuenteDatosBean de fuente de datos
    - Guardar fuente de datos usando XMLAnalisisMaker

- Solicitar conexión a la fuente de datos configurada
- Obtener fuente de datos

- Obtener tablas del proyecto en lista de tablas
- Por cada tabla en lista de tablas
    - Crear BETabla en base a datos de tabla
    - Generar BETablaBean de tabla
    - Guardar tabla usando XMLAnalisisMaker

- Obtener campos de tabla para la lista de tablas
    - Crear BECampo en base a datos del campo
    - Generar BECampoBean de campo
    - Guardar campo usando XMLAnalisisMaker

- Si es necesario obtener datos de tabla
    - Por cada tabla en la lista de tablas
        - Obtener uno a uno los registros correspondientes para la tabla.
        - Por cada registro en lista de tablas
            - Crear BECampo en base a datos del registro
            - Generar BECampoBean de dimensión
            - Cargar en memoria (o en caché) los datos obtenidos.

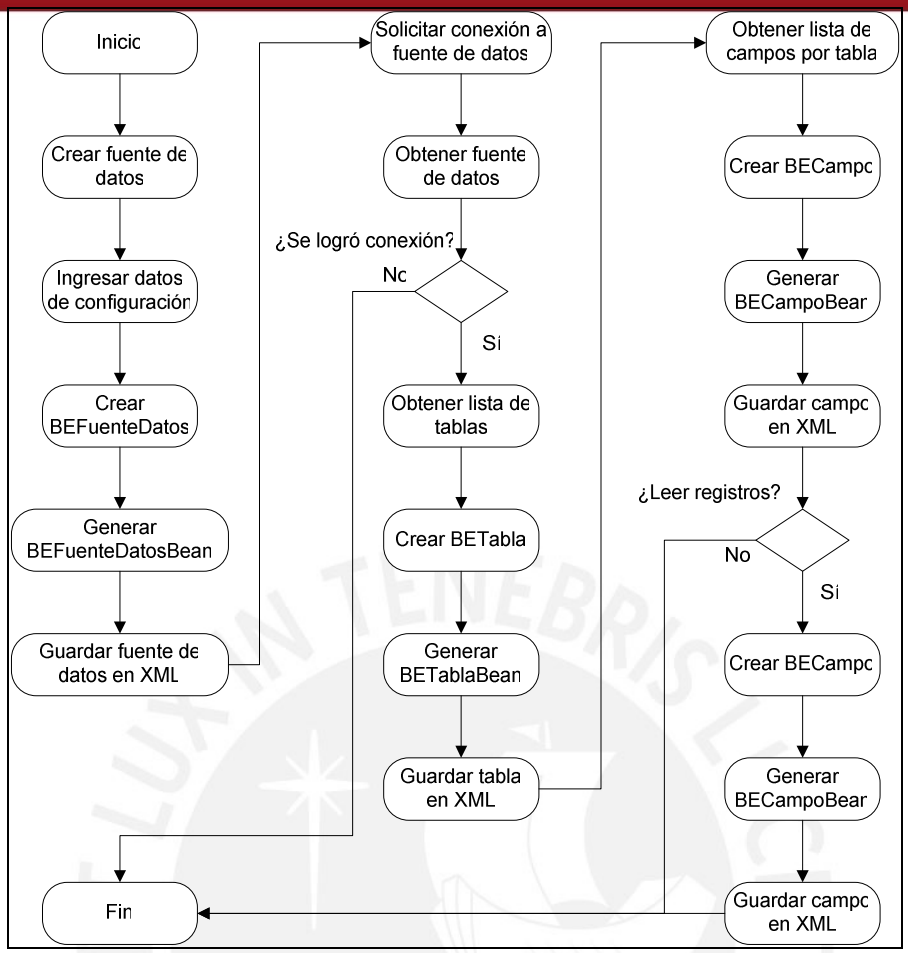


Figura 3.19: Algoritmo: Conexión a fuente de datos

3.5.2. Creación y programación de job

Este algoritmo permite la creación y programación de un *job* en el módulo de extracción para ello se tendrá en cuenta la solicitud de los parámetros adecuados. Los pasos del algoritmo son los siguientes:

Se seleccionó “Crear *job*”:
- Crear objeto BEJob.
- Ingresar datos de configuración del *job* a crear.
- Guardar el *job* usando BEControlXML

Se seleccionó “Programar Job”
- Crear objeto BEProgramacion.
- Ingresar datos de la configuración de la programación a crear.
- Calcular primera ejecución de la programación del *job*, se procede a hallar el tiempo que falta, tomando como base la fecha de creación y a la configuración dada, según:
    - Si es Diaria, se procede a calcular la diferencia de días entre el día actual y el primer día programado.
    - Si es semanal, se halla la diferencia de días entre el día actual, y el día escogido para la ejecución.
    - Si es mensual, primero se verifica si la primera ejecución se realiza este mes o el siguiente, luego se agrega la diferencia de días de acuerdo al día

del mes escogido para la ejecución.
    - Si es anual, primero se verifica si la primera ejecución se realiza este año o el siguiente, luego se agrega la cantidad de meses y los días de acuerdo a la fecha escogida para la ejecución.
- Guardar la programación usando BEControlXML

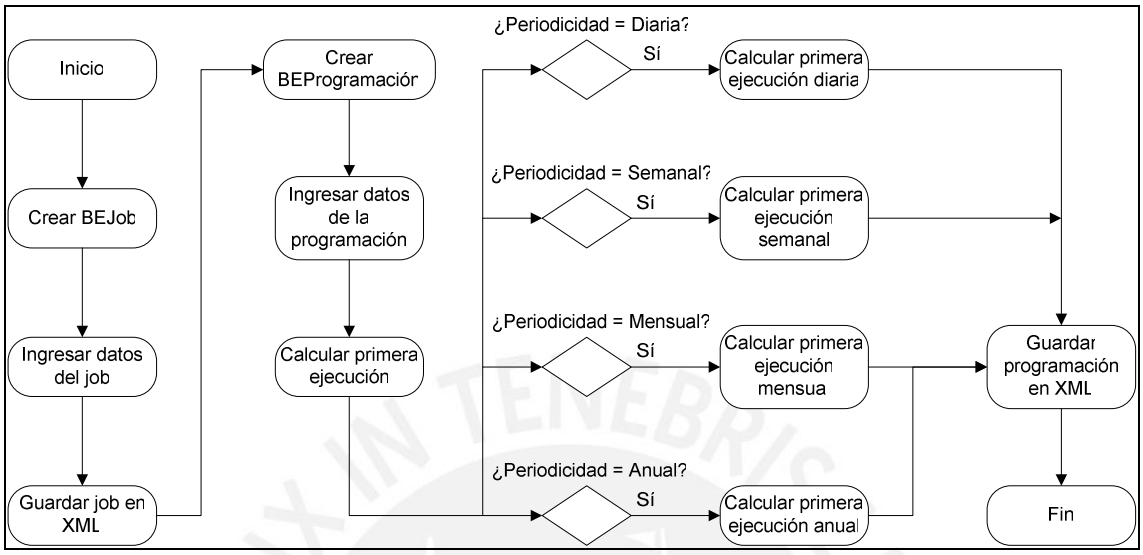


Figura 3.20: Algoritmo: Creación y programación de job

3.5.3. Ejecutar job

Este algoritmo se encarga de la ejecución de los *jobs* programados en el sistema.

- Leer todas las programaciones que existen en el sistema.
- Para cada programación verificar si la fecha próxima de ejecución coincide o es inferior a la fecha actual, si es así proceder a ejecutar el *job* de la programación.
- Para cada *job*:
    - Buscar sus respectivos paquetes y el orden de ejecución.
    - Para cada uno de la paquetes del *job* en ejecución:
        - Obtener sus respectivos componentes, orden de ejecución y fuente de datos configuradas.
        - Para cada uno de los componentes:
            - Obtener los objetos pasivos y activos respectivos.
            - En caso el objeto es pasivo, se realiza la conexión y se obtienen los datos correspondientes. Caso contrario, el objeto es activo, en este caso se revisan uno a uno los componentes internos configurados como filtros, reglas de estandarización, etc, y se procede a la ejecución de cada uno de estos paso a paso.
- Para cada una de las acciones realizadas, guardar en el Log de ejecución.
- Al terminar la ejecución, calcular la siguiente fecha de ejecución del *job*, según su programación:
    - Si es Diaria, se procede a calcular la diferencia de días entre el día de ejecución (actual) y el siguiente día programado.
    - Si es semanal, hallar la diferencia de días entre el día de ejecución (actual), y el día escogido para la ejecución, tomando en cuenta la separación entre semanas dada por la programación semanal.
    - Si es mensual, agregar el número de meses según la separación entre

meses dada por la programación mensual.
- Si es anual, agregar el número de años según la separación entre años dada por la programación anual.

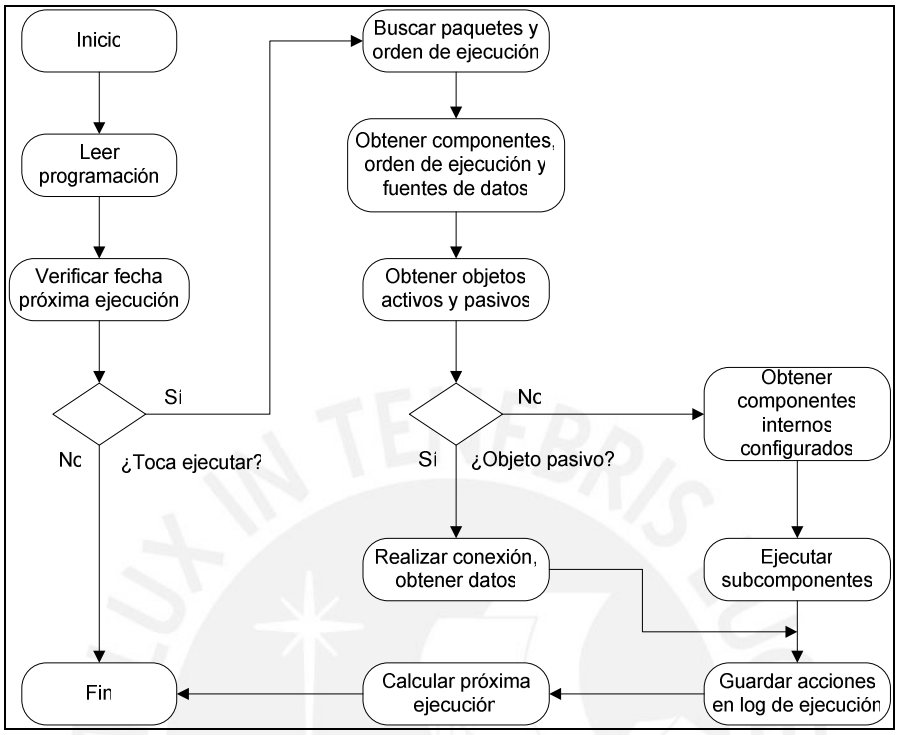


Figura 3.21: Algoritmo: Ejecutar job

3.5.4. Ejecutar filtro

Este algoritmo consiste en la ejecución del subcomponente de filtro dentro del flujo de transformación. Como resultado de la ejecución del filtro se generan archivos intermedios que serán utilizados en los subcomponentes que continúan en la configuración del flujo de transformación. Los pasos del algoritmo son los siguientes:

- Obtener orígenes de datos.
- Para cada origen de datos:
  - Obtener lista de sentencias del componente.
  - Para cada sentencia del componente:
    - Si la sentencia involucra al origen de datos
      - Agregar sentencia al Filtro para el origen de datos.
  - Si existe Filtro para el origen de datos
    - Crear archivo intermedio.
    - Si la Fuente de datos es una BD
      - Abrir conexión a la fuente de datos.
      - Ejecutar Query (Filtro) en la fuente de datos.
      - Para cada registro obtenido como resultado:
        - Agregar registro al archivo intermedio
      - Se cierra la conexión a la fuente de datos.
    - Caso contrario :
      - Abrir archivo plano para lectura

- Leer registro del archivo
- Para cada registro del archivo:
  - Verificar que el registro cumpla con el filtro
  - Si cumple:
    - Agregar registro al archivo intermedio.
- Cerrar Archivo plano.

- Cerrar archivo intermedio
- Agregar el archivo a Lista Archivos Filtrados que se pasarán a los demás subcomponentes.

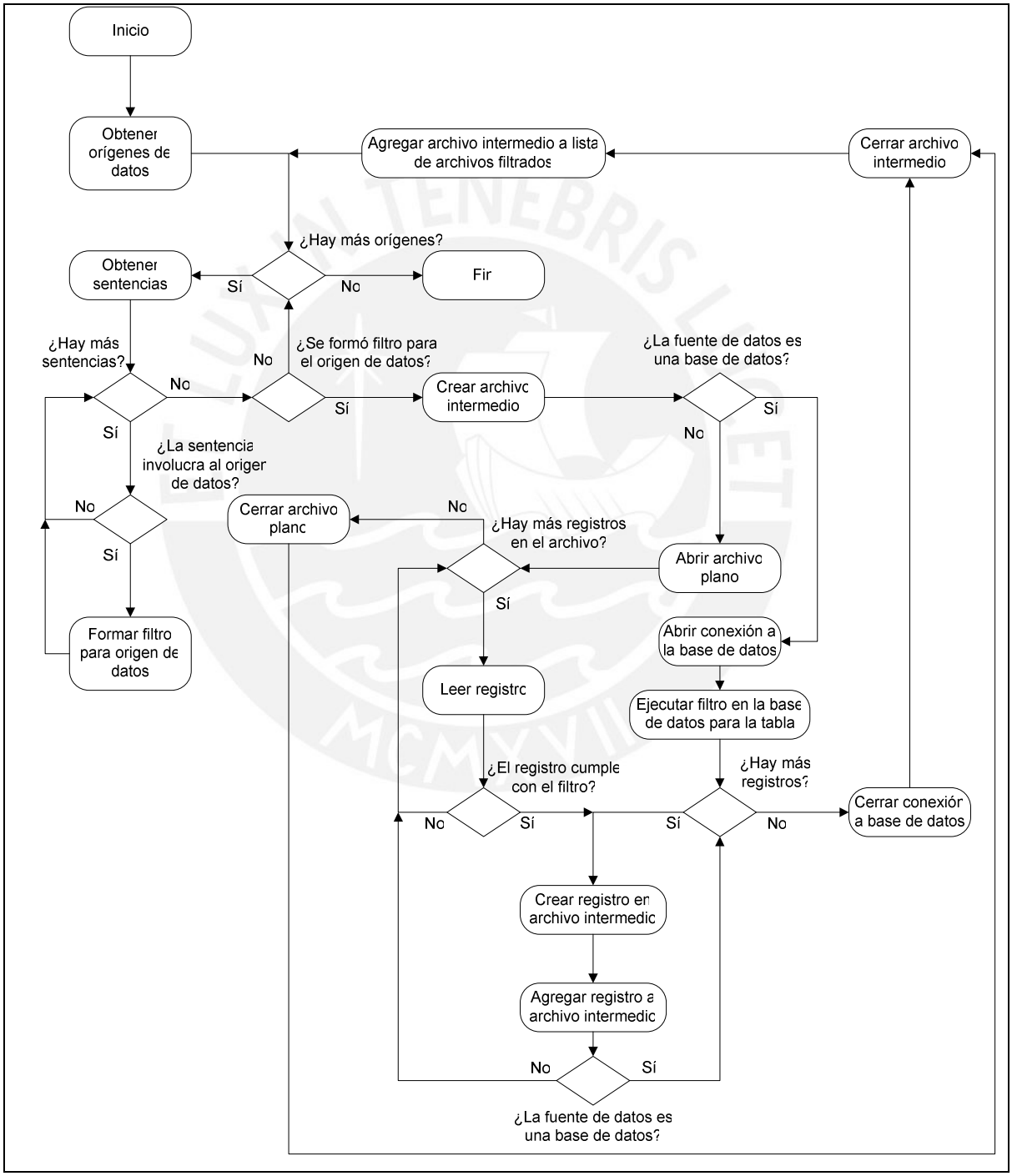


Figura 3.22: Algoritmo: Ejecutar filtro

3.5.5. Ejecutar transformación

Este algoritmo consiste en la ejecución del subcomponente de transformación dentro del flujo de transformación. Como resultado de la ejecución de la transformación se genera un archivo intermedio que será leído en el subcomponente que continúa en la configuración del flujo de transformación. Los pasos del algoritmo son los siguientes:

- Revisar si existen archivos intermedios resultado del subcomponente de filtro. Si los hay entonces cargar los archivos a memoria para poder leer de ellos.
- Revisar si hay más orígenes de datos en el flujo para este componente, aparte de los que se obtuvieron del filtro. Si los hay, se abre una conexión a la fuente de datos respectiva a cada origen.
- Obtener las sentencias de la transformación y convertirlas a objetos sentencia.
- Crear el archivo destino de la transformación.
- Ordenar los orígenes de datos de acuerdo al número de referencias en *constraints*
- Para cada registro del primer origen de datos:
    - Crear un conjunto de datos.
    - Verificar si el registro actual cumple los *constraints*.
    - Si cumple:
        - Agregar el valor al conjunto de datos actual.
        - Si ya se completó el conjunto de datos:
            - Aplicar las sentencias al conjunto de datos.
            - Guardar el registro resultado en el archivo intermedio.
        - Caso contrario:
            - Recorrer los registros del siguiente origen de datos para completar el conjunto.
- Cerrar el archivo resultado
- Cerrar los archivos intermedios de filtro
- Borrar los archivos intermedios de filtro
- Cerrar las conexiones a fuentes de datos

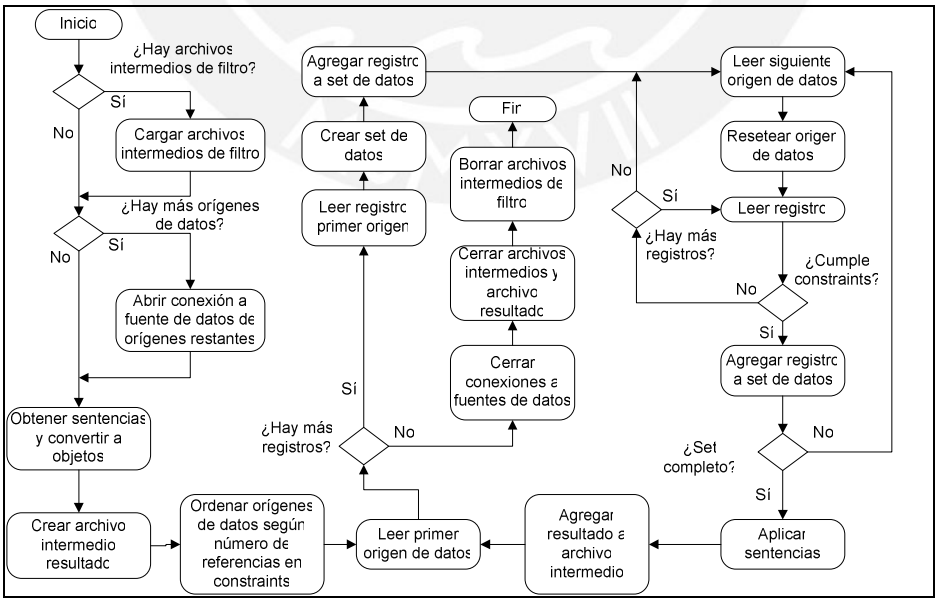


Figura 3.23: Algoritmo: Ejecutar transformación

3.5.6. Ejecutar estandarización

Este algoritmo consiste en la ejecución del subcomponente de estandarización dentro del flujo de transformación. Como resultado de la ejecución de la estandarización se genera un archivo intermedio que será leído para la carga de datos al destino. Los pasos del algoritmo son los siguientes:

- Revisar si existen archivos intermedios resultado de la transformación. Si los hay entonces se utilizará los datos del archivo para procesar la estandarización.
- Si no existen archivos intermedios, entonces revisar si existe un origen de datos desde donde procesar la estandarización. Si el origen de datos es correcto se leerá los datos de esta fuente.
- Si no se tiene archivos intermedios disponibles ni orígenes de datos correctos entonces no es posible la estandarización.
- Obtener la lista de sentencias de estandarización
- Crear un nuevo archivo intermedio para almacenar los datos procesados
- Obtener un registro de datos para estandarizar
- Recorrer todos los campos del registro y hacer corresponder con cada una de las sentencias de la lista de sentencias de estandarización
- Para cada campo del registro:
    - De la sentencia asociada obtener la operación o función de estandarización
    - Aplicar la operación obtenida sobre el campo correspondiente
    - Añadir campo estandarizado a un nuevo registro
    - Agregar el registro estandarizado en el nuevo archivo intermedio
- Cerrar los archivos abiertos
- Cerrar las fuentes de datos abiertas, si fuera necesario.

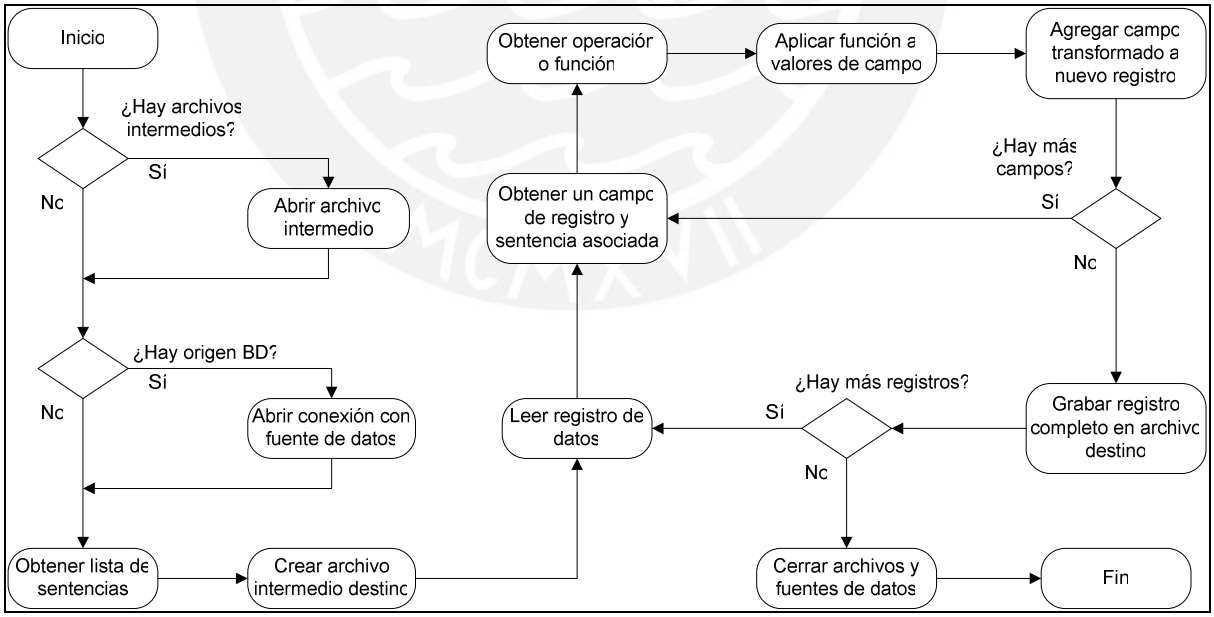


Figura 3.24: Algoritmo: Ejecutar estandarización

3.5.7. Ejecutar carga de datos

Este algoritmo consiste en la ejecución de la carga de datos al final del flujo de transformación. Como resultado de la ejecución de la carga de datos se tendrán los datos transformados almacenados en la fuente de datos destino según la configuración del flujo de transformación. Los pasos del algoritmo son los siguientes:

- Revisar si existen archivos intermedios de datos para cargar al destino. Si existiesen estos se utilizarán como fuente de datos.
- Si no existen archivos intermedios, revisar si existen fuentes de origen de datos desde donde cargar los datos al destino. Si existen se leerá los datos desde esta fuente.
- Si no se tiene archivos intermedios disponibles ni orígenes de datos correctos entonces no es posible la carga.
- Revisar la conexión con la fuente de datos destino para verificar si es válida, en caso sea incorrecta no será posible la carga.
- Obtener un registro de datos para estandarizar
- Para cada registro realizar lo siguiente:
    - Obtener el identificador o llave primaria del registro
    - Verificar si el registro ya existe en la tabla destino. Si el registro no existe entonces insertar el registro actual en la tabla destino. Caso contrario:
    - Actualizar el registro de la tabla destino con los valores de los campos del registro actual.
- Cerrar los archivos intermedios, si fuera necesario
- Cerrar las conexiones de fuente de datos abiertas.

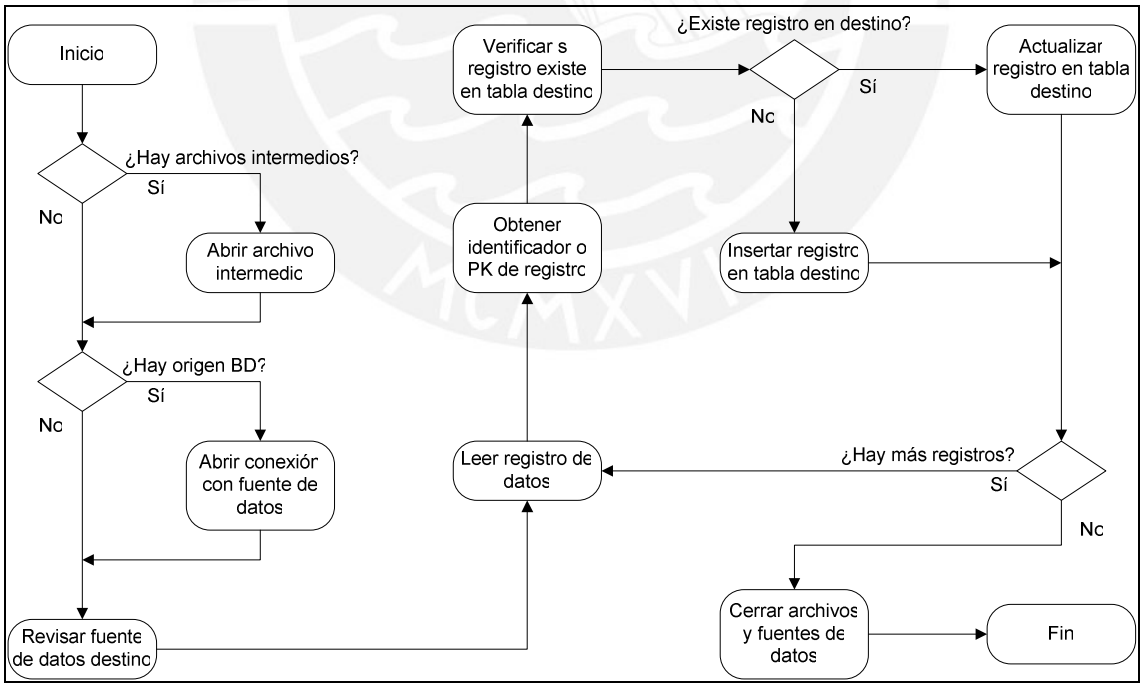


Figura 3.25: Algoritmo: Carga de datos

3.5.8. Conversión archivo plano

Este algoritmo consiste en convertir un archivo plano, de acuerdo con la configuración ingresada por el usuario, a objetos propios de la herramienta que puedan ser utilizados para construir el flujo de transformación. Los pasos del algoritmo son los siguientes:

- Abrir archivo plano para lectura.
- Para cada registro del archivo
    - Para cada campo del archivo
    - Si el campo es Primary Key
        - Validar que el campo no sea nulo
        - Si es nulo
            - Fin de *parser* – Archivo Inválido
        - Caso contrario:
            - Validar unicidad de datos
            - Si el dato no es único
                - Fin de *parser* – Archivo Inválido.
    - Caso contrario:
        - Validar nulidad del dato.
        - Si el campo no es nulo
            - Validar Tipo de dato
            - Si es válido:
                - Validar tamaño
                - Si es válido
                    - Si el dato contiene decimales
                        - Validar decimales.
                        - Si no es válido
                            - Fin de *parser*-Archivo Inválido.
                - Caso contrario
                    - Fin de *parser* – Archivo Inválido
            - Caso contrario
                - Fin de *parser* – Archivo Inválido.
- Si se llegó al fin de archivo: Archivo Válido.
- Cerrar Archivo plano.
- Guardar configuración de campos.

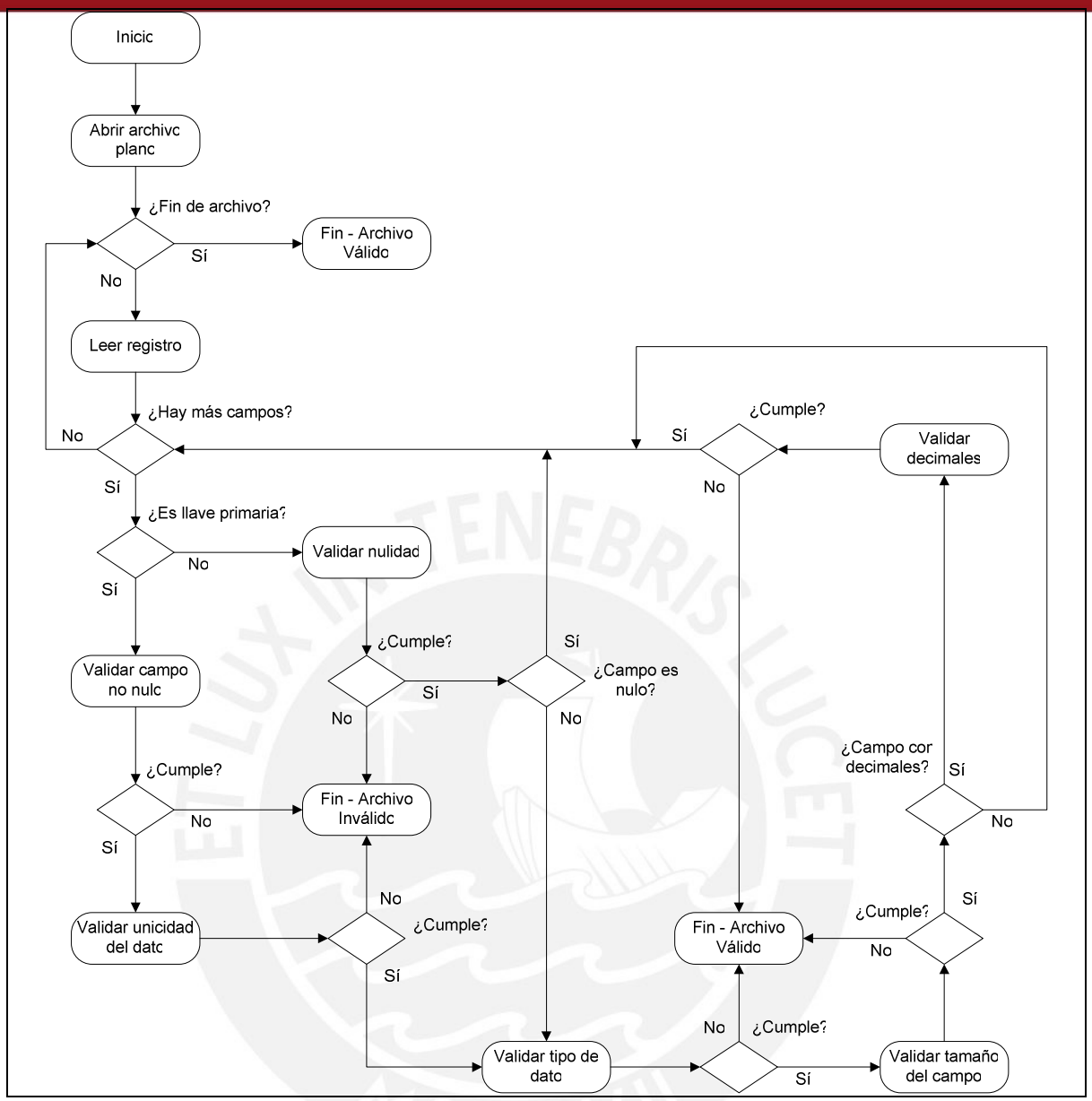


Figura 3.26: Algoritmo: Parsear archivo plano

4. OBSERVACIONES, CONCLUSIONES Y RECOMENDACIONES

4.1. Observaciones

- En la actualidad existen diversas herramientas que ofrecen soluciones de Inteligencia de Negocios, la mayor parte de ellas orientadas a las grandes organizaciones, esta herramienta está dirigida a medianas organizaciones de forma que puedan explotar al máximo su información para su crecimiento.
- El proyecto que se presenta como trabajo de tesis ha pasado por varias etapas de evolución. En un inicio surgió como una idea de un grupo de profesores de la universidad. Con miras a materializar la idea, se convocó a un grupo de estudiantes para formar parte del equipo de desarrollo. Luego de formado el equipo, se concretó una serie de sesiones de capacitación sobre el tema de Inteligencia de Negocios para que el equipo tenga la idea de lo que se buscaba lograr con el proyecto (verano 2004). El producto fue tomando forma en una versión preliminar al ser trabajado por el equipo de desarrollo como parte del curso de Desarrollo de Programas 1 (primer semestre 2005). A inicios del 2006 se comenzó a definir los alcances de la versión definitiva del producto. En ese momento, el proyecto fue tomado como tema de tesis para los integrantes del equipo de desarrollo. Finalmente, el desarrollo del producto se ha prolongado por todo el año 2006 y se tiene planeado presentarlo como un producto comercial en el primer semestre del 2007.
- Mediante el presente trabajo de tesis no sólo se ha desarrollado una solución integral de Inteligencia de Negocios, sino que para el mismo se ha tenido en cuenta una investigación completa sobre los conceptos, metodologías y técnicas que la constituyen.

- La herramienta que se ha diseñado puede alimentarse no sólo de una base datos especializada como la de Oracle, sino que puede recibir también datos de archivos de texto, archivos de hoja de cálculo Microsoft Excel y archivos XML.
- La solución de Inteligencia de Negocios presentada es una herramienta útil, flexible, de fácil manejo, coherente con el ambiente empresarial actual, que puede ser fácilmente utilizada en diferentes organizaciones.
- La aplicación es versátil en el sentido que no está sujeta a un estándar específico sino que da libertad, a la empresa que la utilice, de poder aplicar sus propios análisis y estructura de orígenes de datos. Esta característica permite un manejo más independiente y personalizado de las funcionalidades. El modelo de ejecución y trabajo dentro de la aplicación está apoyado en criterio de simplicidad y facilidad de manipulación.
- El presente trabajo de tesis fue realizado por tres integrantes, en interacción constante con los otros dos módulos que constituyen el producto integral. Este trabajo es punto de partida para la tesis: “Construcción y pruebas de una herramienta de desarrollo de soluciones para inteligencia de negocios – Módulo de Extracción”, que se basó en el análisis y diseñado presentado para llevar a cabo la solución descrita. El objetivo final es enlazar el módulo desarrollado a una Solución Integral de Inteligencia de negocios, que permita alimentarse y entregar datos a los otros dos módulos que conforman el paquete de aplicaciones.

4.2. Conclusiones

- Las herramientas usadas para la implementación del producto cumplieron con las expectativas del equipo de desarrollo para el proyecto, brindando facilidades que agilizaron el proceso y permitieron una integración rápida y sin contratiempos de los módulos del producto.
- La temprana definición del estándar de pantallas y navegación permitió al equipo construir un conjunto de herramientas consistente y que permite al usuario ubicarse rápidamente al pasar de un módulo al siguiente. Al realizar una prueba con un usuario nuevo, este demoró menos de 15 minutos en acostumbrarse al manejo de la herramienta y ubicación de las funcionalidades principales para las tareas básicas de la herramienta como son: crear un *job*, crear un paquete, conectar a una fuente de datos y formar el flujo de transformación.
- La herramienta construida es escalable a futuro, tanto a nivel de nuevos tipos de fuentes de datos para realizar las tareas de *ETL* como a nivel de páginas *JSF* para poder mostrar la herramienta en diferentes idiomas, de acuerdo al equipo cliente. Para lograr este objetivo se hizo uso de técnicas de diseño que se listan en los

siguientes 3 puntos.

- El hacer uso de un esquema de herencia y polimorfismo en los objetos activos y pasivos, así como también en los objetos de dibujo permitieron construir fácilmente las diferentes clases de fuentes de datos derivando de un tipo común. De esta forma el mantenimiento y mejoras futuras de la aplicación no son una tarea ardua sino por el contrario sencillas.
- En el caso de las conexiones a motores de bases de datos, se utilizó un esquema que permite ampliar los tipos de motores a los cuales puede conectarse la herramienta de forma sencilla. Sólo es necesario contar con el *driver* adecuado para el motor de datos a agregar y realizar pequeños cambios en el controlador de fuentes de datos. Esto le da a la herramienta una cualidad de alto potencial de escalabilidad, pues permitiría, con modificaciones menores, la inclusión de nuevas formas de manejo de datos que puedan surgir en el futuro.
- Se definió el uso de archivos *bundle* para contener el texto de las páginas JSF, tanto del título como en el cuerpo y componentes de las mismas. De esta forma se logró separar el contenido de las páginas del código *JSF* lo que permite la internacionalización de la herramienta. Es decir, mediante la inclusión de varios archivos *bundle*, uno para cada idioma, será posible mostrar la herramienta en varios idiomas sin necesidad de hacer cambios en el código *JSF*.

4.3. Recomendaciones

- Dado que la aplicación diseñada es web es necesario proteger el servidor de ataques externos, ya sea con firewalls o cualquier tecnología que exista actualmente, de manera que se asegure el funcionamiento del servidor las 24 horas del día, sin ningún tipo de interrupciones.
- La herramienta ha sido diseñada para ser de fácil uso, sin embargo, se considera necesario contar con la ayuda completa, para que se pueda usar el máximo de sus potencialidades.
- En el futuro, si se necesita ampliar la funcionalidad de la herramienta, como por ejemplo, agregar más tipos de fuentes de datos, lo más conveniente será tomar como base las clases genéricas existentes. Creando las nuevas fuentes de datos como clases que heredan de las clases genéricas se logrará aprovechar al máximo la implementación existente, sobreescribiendo sólo los métodos específicos para el funcionamiento del nuevo tipo de fuente de datos.

BIBLIOGRAFÍA

1. [ALM 1999] Maria Sueli Almeida, Missao Ishikawa, Joerg Reinshmidt, Torsten Roeber. 1999. Getting Started with Data Warehouse and Business Intelligence. IBM Technical Support Organization.

2. [KIM 2002] Ralph Kimball, Margy Ross. 2002. The Data Warehouse Toolkit. Second Edition, Wiley Computer Publishing.

3. [SAG 2006] http://www.sagent.com.ar/stecno.htm. Sagent Technology home page.

4. [MST 2006] http://www.microstrategy.com. MicroStrategy home page.

5. [BUS 2006] http://www.businessobjects.com. Business Objects home page.

6. [COG 2006] http://www.cognos.com. Cognos home page.

7. [SUN 2006] http://www.sunopsis.com/corporate/index.htm. Sunopsis home page.

8. [MSTG 2006] http://es.ascential.com/productos/datastage.html. DataStage home page.

9. [BOU 2005] http://www.rpbourret.com/xml/XMLAndDatabases.htm. Ronald Bourret. 1999-2005. XML and Databases. Ronald Bourret home page.

10. [JDO 2006] http://www.jdom.org/. JDOM Project home.

11. [CAS 2006] http://www.castor.org/ Castor Project home.

12. [RMI 2006] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp. Remote Method Invocation home.

13. [REA 2005] http://www.r-software.com/Doctos/patrones.pdf. Real Software S.A. de C.V. 2005. Diseño de aplicaciones internet usando los patrones de diseño J2EE.

14. [KRU 2000] Phillippe Krutchen. 2000. The Rational Unified Process: An Introduction. Segunda edición, Addison-Wesley Professional.

15. [IFP 1999] The International Function Point User Group (IFPUG). 1999. Function Point Counting Practices Manual Release 4.1. USA.

16. [COC 2002] http://sunset.usc.edu/research/COCOMOII/. Cocomo II. 2002.

ANEXOS