Universidad de las Ciencias Informáticas

FACULTAD 6

Título: *Componente de transformación del Pentaho Data Integration para añadir constantes a partir de valores almacenados en bases de datos.*

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor:

Alian Ortega Herrera

Tutores:

Msc. Yanet Villanueva Armenteros

Lic. Lisdán Rodríguez Pérez

Co-tutora:

Ing. Doris Medina Mustelier

La Habana, junio de 2014.

"Año 56 de la Revolución"

"El futuro tiene muchos nombres. Para los débiles es lo inalcanzable. Para los temerosos, lo desconocido. Para los valientes es la oportunidad." *Victor Hugo*

DECLARACIÓN DE AUTORÍA

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

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

_____

Alian Ortega Herrera

Autor

_____

MsC. Yanet Villanueva Armenteros

Tutora

_____

Lic. Lisdán Rodríguez Pérez

Tutor

_____

Ing. Doris Medina Mustelier

cotutora

DATOS DE CONTACTO

Autor:

Alian Ortega Herrera

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: aortega@estudiantes.uci.cu

Tutora:

MsC. Yanet Villanueva Armenteros

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: villanueva@uci.cu

Tutor:

Ing. Lisdán Rodríguez Pérez

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: lisdanrp@uci.cu

Co-Tutora:

Ing. Doris Medina Mustelier

Universidad de las Ciencias Informáticas, La Habana, Cuba

Email: dmedina@uci.cu

AGRADECIMIENTOS

Quisiera agradecer en primer lugar a mis abuelos ***Angel*** *y* ***Belkis****, ellos han sido mis segundos padres, mi abuelo donde quiera que esté sé que debe estar orgulloso de mi, gracias mima y papi por quererme tanto.*

A mi mamá ***Gladys,*** *por ser la mejor de todas las madres para mí, gracias por darme la vida y por todo el sacrificio que has hecho por mí.*

A mis tías ***Niurkis****,* ***Evelin*** *y* ***Leidis,*** *sin su apoyo todo esto hubiera sido imposible, han sido como madres durante todo esto tiempo.*

A mis hermanos ***Osvaldo*** *y* ***Osmany,*** *que aunque son un poco mal criados yo los quiero mucho y sé que ellos me quieren mucho a mí.*

A mi hermana ***Yéssica*** *y mi* ***sobrinito****, que aunque no he tenido mucho contacto con ellos, siempre los tengo presente.*

A todos mis primos, Rosley, Rosilileidis, Carlito, Lisbet, Adrián, Daimara y Daisara, que también han influido mucho en mi vida.

A mi novia ***Taimí,*** *por dejarme entrar en su vida, por pasar buenos y malos momentos desde el día que nos conocimos, por quererme y apoyarme tanto cuando pensé que todo estaba perdido, espero que esta bella relación perdure por mucho tiempo.*

A todos los que alguna vez fueron mis compañeros de aula, en especial a los que tengo en estos momentos.

A todos mis amigos, que han compartido conmigo durante estos 5 años, también a los que ya no están por un motivo u otro, todos han aportado mucho a mi vida y a mi formación. Sé que se me quedarán algunos, pero si quisiera darles un agradecimiento especial a ***Yadián,*** *el primer amigo que tuve en la universidad y que ya lo considero mi hermano, al* ***Duque, Yusniel, Jorge Luis, Miguel, Rolando, Yoelkis, Maikel, Reinaldo, Roilán, Luis Alberto, Juan Miguel, Eduar, Meyker*** *mi hijo,* ***Dianeyis, Dunia,*** *a mis nuevos*

compañeros de apartamento, más conocidos como el piquete del dominó, ***Bofil*** *el animal,* ***Chávez, El gato, El tuna, Roger*** *el déspota y por último y no menos importantes a* ***Mauricio*** *que ha sido como un padre para mí y a* ***Fabré*** *el otro hijo de Mauricio, que es como un hermano para mí.*

A mis tutores ***Doris*** *y* ***Lisdán,*** *que sin sus consejos, críticas, llamadas de atención y su ayuda, no hubiera podido alcanzar mi sueño de formarme como profesional.*

A los miembros del tribunal y a mi oponente por sus críticas constructivas, que aportaron mucho en la realización de la tesis.

DEDICATORIA

La presente tesis va dedicada especialmente a mi difunto abuelo Angel Herrera, a mi tío con el mismo nombre que ya no está físicamente, a mi abuela Belkis, a mi mamá, a mi novia y a toda mi familia en general, todos han sido muy especiales para mí y han ayudado y contribuido a mi formación como profesional, gracias por su apoyo incondicional.

RESUMEN

La presente investigación se enmarca en la implementación e integración de un componente a la herramienta Pentaho Data Integration, cuyo objetivo es permitir añadir constantes a partir de campos almacenados en base de datos. Para el desarrollo del componente, se realizó un estudio detallado del PDI, sus ventajas, propiedades, así como la manera de integrarle nuevos componentes y se definieron las herramientas y tecnologías a utilizar en el desarrollo del mismo. La metodología utilizada durante la investigación fue la OpenUp y como lenguaje de programación se utilizó Java. Se diseñaron e implementaron las clases del componente, aplicando las buenas prácticas del patrón arquitectónico basado en plugin y de los patrones de diseño. Fueron realizadas varias pruebas, con el objetivo de comprobar el funcionamiento y la calidad de este nuevo componente. Finalizado el proceso de desarrollo, se obtiene un nuevo componente capaz de corregir las deficiencia existente en el anterior (Añadir constantes), permitiendo añadir constantes a partir de campos almacenados en base de datos. Además se obtiene la documentación de los resultados alcanzados, que será de gran ayuda para próximas versiones del mismo.

Palabras claves:

Pentaho Data Integration, componente, constantes, plugin

ABSTRACT

This research is part of the implementation and integration of a component to the Pentaho Data Integration (PDI) tool, which aims to enable add constants from fields stored in database. For the development of component, a detailed study was conducted, its advantages, properties, and how to integrate you new components was performed and the tools and technologies were defined for use in the development. The methodology used for the research was OpenUp and Java programming language was used. He designed and implemented the component classes, applying the best practices of the architectural pattern based on plugin and design patterns. Several tests were performed with the aim of checking the operation and the quality of the new component. Terminated the development process, is obtained a new component capable of correcting existing deficiencies in the above (Add constant), allowing to add constants from fields stored in database. Further documentation of the results achieved, which will help for future versions thereof is obtained.

Keywords:

Pentaho Data Integration, component, constant, plugin.

ÍNDICE

ÍNDICE

Índice de figuras

Índice de tablas

INTRODUCCIÓN

El hombre con el transcurso de los años, ha alcanzado un gran desarrollo científico y tecnológico. El internet, la computación, los dispositivos móviles, los datos científicos y el uso de diferentes informaciones, han potenciado el desarrollo de la ciencia, la tecnología y otras áreas del conocimiento en el mundo. En la actualidad la cantidad de información histórica almacenada y generada en forma digital crece continuamente. Según la compañía IBM[1], solo en los últimos dos años se ha generado el 90% de la información total existente en el mundo. Este gran volumen de información trae consigo problemas para el hombre, ya que esta se hace cada vez más difícil de almacenar y de consultar para realizar análisis o tomar decisiones.

Cuba no está exenta de las dificultades ocasionadas por el gran volumen de información que se ha generado en la actualidad. La Universidad de las Ciencias Informáticas (UCI), cuenta con varios centros de desarrollo de software y entre ellos se encuentra el Centro de Tecnologías de Gestión de Datos (DATEC), que a su vez contiene el departamento Almacenes de Datos. Este departamento se dedica al análisis de los datos y es el encargado entre otras tareas, de realizar aplicaciones como los almacenes de datos, que automaticen el proceso de consulta de información histórica o actual de cualquier empresa o entidad, con el objetivo de contribuir a la toma decisiones importantes con mayor facilidad y seguridad.

Una de las fases para la obtención de un almacén de datos, es el proceso de Extracción, Transformación y Carga (ETL, por sus siglas en inglés). En esta fase el departamento Almacenes de Datos utiliza la herramienta Pentaho Data Integration (PDI), la misma fue creada para cubrir las necesidades en la integración de datos, que consiste en integrar la información, con el objetivo de brindársela al usuario. La herramienta proporciona un gran número de beneficios, dentro de ellos se puede mencionar especial facilidad de uso gracias a una interfaz muy intuitiva. Además el uso de PDI evita grandes cargas de trabajo manual, frecuentemente difícil de mantener y de desplegar. La herramienta contiene diversos componentes, cada uno con diferentes funcionalidades necesarias para el tratamiento de los datos.

[1] International Business Machines es una empresa multinacional estadounidense de tecnología y consultoría con sede en Armonk, Nueva York.

Existen componentes de entradas, de transformación, de búsquedas, de salida, entre otros no menos importantes.

Dentro de los componentes de transformación se encuentra Añadir constantes, uno de los originales del PDI, ya que existen otros que han sido implementados posteriormente, para cubrir necesidades que se han presentado en el trabajo con la herramienta. El mismo tiene como objetivo añadir constantes al flujo de datos, las cuales son utilizadas durante toda la transformación. En el uso del componente se ha presentado una deficiencia, ya que no permite conectarse a base de datos. En ocasiones en el diseño del flujo de datos se han presentado situaciones donde los valores de estas constantes necesarias para trabajar en algún momento específico, se encuentran almacenadas en bases de datos, por lo que se debe acceder de forma manual a la misma, para poder chequear el valor y el tipo de la constante. Otra vía sería utilizar otros componentes como consultas SQL, lo que trae como consecuencia que el trabajo se haga más engorroso y complicado, demorando la obtención de los resultados y arrojando dificultades como errores a la hora de chequear el tipo y el valor de la constante por parte de los especialistas.

Por todo lo antes expuesto se plantea como **Problema de la investigación**: ¿Cómo añadir constantes a partir de valores almacenados en bases de datos a través de la herramienta Pentaho Data Integration?

La investigación plantea como **objeto de estudio** las herramientas de integración de datos, enmarcado en el **campo de acción** componente de transformación para la herramienta Pentaho Data Integration.

Se define como **objetivo general** de la investigación: Desarrollar un componente de transformación para la herramienta Pentaho Data Integration, que permita añadir constantes a partir de valores almacenados en bases de datos.

Para llevar a cabo el desarrollo del objetivo general se determinan las siguientes **preguntas científicas:**

¿Cómo se puede integrar un nuevo componente a la herramienta Pentaho Data Integration, que permita añadir constantes desde campos almacenados en base de datos?

¿Cuáles son las características que debe presentar el componente, para que permita añadir constantes desde campos almacenados en base de datos?

¿Cómo realizar el proceso de desarrollo del componente, para lograr añadir constantes desde campos almacenados en base de datos?

¿El componente puede añadir constantes desde campos almacenados en base de datos?

Para cumplimentar el objetivo general se definen los siguientes **objetivos específicos:**

1. Realizar un estudio detallado de la herramienta Pentaho Data Integration, para conocer cómo integrar un nuevo componente que permita añadir constantes a partir de valores almacenados en base de datos.
2. Fundamentar la selección de la metodología, herramientas y tecnologías a utilizar para el desarrollo del componente que permita añadir constantes a partir de valores almacenados en base de datos.
3. Realizar el análisis y diseño del componente que permita añadir constantes a partir de valores almacenados en bases de datos.
4. Implementar el componente que permita añadir constantes a partir de valores almacenados en bases de datos.

Para cumplir con los objetivos específicos se planifican las siguientes **tareas de investigación:**

1. Revisión bibliográfica de la documentación técnica y la arquitectura de Pentaho Data Integration referida a la extensión de la herramienta, para fundamentar acerca de su estructura y funcionamiento.
2. Selección de la metodología, herramientas y tecnologías a utilizar en el desarrollo del componente para crear constantes a partir de valores almacenados en bases de datos, que permita la organización y el diseño de la investigación, así como su realización.
3. Realización del modelado de los conceptos fundamentales, para lograr una mayor comprensión de la solución.
4. Realización del modelado de las entidades y eventos del dominio, para realizar el diseño de la investigación, usando la metodología, herramientas y tecnologías escogidas.
5. Implementación del componente para crear constantes a partir de valores almacenados en bases de datos, para cumplir con los requisitos definidos en el negocio.

6. Definición de las pruebas a realizar al componente, que permita guiar las pruebas del sistema.
7. Realización de pruebas al componente para crear constantes a partir de valores almacenados en bases de datos, para verificar el funcionamiento del mismo.

Para la realización de las tareas fueron utilizados métodos de investigación teóricos y empíricos.

Métodos teóricos

Histórico-Lógico: fue utilizado con el objetivo de estudiar todos los elementos presentes en el desarrollo de otros componentes para la herramienta Pentaho Data Integration.

Análisis-Síntesis: el uso de este método permitió analizar informaciones recopiladas de bibliografías previamente consultadas, así como llegar a concretar aspectos y arribar a conclusiones que ayuden al desarrollo de la investigación, mediante el análisis de algunas informaciones adquiridas con anterioridad.

Métodos empíricos

Entrevista: este método permitió realizar varias entrevistas al cliente, lo cual posibilitó definir de forma correcta, todas las necesidades que se deben cumplir con el desarrollo de la investigación.

Estructura del documento

Capítulo 1: Fundamentación teórica.

Este capítulo contiene la fundamentación teórica del trabajo, en el mismo se realiza un análisis de los principales conceptos relacionados con la herramienta Pentaho Data Integration, así como la justificación de la metodología escogida y las herramientas utilizadas durante la investigación.

Capítulo 2: Análisis y diseño del componte.

En este capítulo se plantea el análisis y diseño de la solución. En él quedan definidas todas las funcionalidades y características que debe cumplir el componente y además se realiza una descripción de su funcionamiento. Se identifican los actores, así como los casos de usos del sistema, agrupando dentro de estos, los diferentes requisitos funcionales. Se realiza el diagrama de clases del diseño abordando acerca de los patrones utilizados, patrones de diseño y patrones arquitectónicos.

Capítulo 3: Implementación y prueba del componte.

En este capítulo se hace referencia a los elementos necesarios utilizados durante la implementación del componente. Se realizó el diagrama de componentes del sistema, mostrando las relaciones que existen entre los diferentes objetos que componen el diagrama. Se definieron los estándares de codificación utilizados y además fueron realizadas las pruebas al componente para comprobar su funcionamiento.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA SOBRE LA INTEGRACIÓN DE NUEVOS COMPONENTES A LA HERRAMIENTA PENTAHO DATA INTEGRATION.

En este capítulo se realiza un estudio sobre los elementos que fundamentan la base teórica, de la investigación. Se hace un análisis de los principales conceptos y se incluye una breve descripción de los lenguajes, las herramientas, tecnologías y metodologías utilizadas para la elaboración del componente.

1.1. Conceptos importantes asociados a la investigación.

Con el objetivo de lograr una mayor comprensión de lo descrito en la investigación, se exponen a continuación varias definiciones de términos que se abordan en la misma, los cuales son de gran relevancia.

1.1.1. Definición de constante.

En términos de programación, una constante es un símbolo que representa el valor específico de un dato determinado. Cuando es definida, puede tomar varios formatos, que depende del tipo de datos del valor que se desea representar. (1) El nombre de las constantes se suele escribir con letra mayúscula en la mayoría de los lenguajes de programación.

Las constantes pueden ser intrínseca o definidas por usuarios, la primera clasificación significa que tiene nombres y valores preestablecidos. Por otra parte las constantes definidas por el usuario, reciben el nombre y el valor que este desee o necesite para su trabajo. Se debe aclarar que el usuario no puede crear una constante con el mismo nombre de una intrínseca, ni declarar dos con el mismo nombre, aunque sí con el mismo valor. Un ejemplo de estas declaraciones de constantes se muestra en la tabla 1:

Tabla 1: Ejemplo de declaraciones de constantes

```
<%
Const MICONSTANTE = "Esto es un string Constante"
Const DIA = 28
Const FECHA_PREDETERMINADA=  #02-28-2000#
%>
```

1.1.2. Definición de componente

Un componente es una pieza de código pre-elaborado, que expone a través de interfaces estándar alguna o varias funcionalidades. Posee una serie de interfaces y de requisitos, y además puede ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio. (2)

Las interfaces de un componente son las encargadas de exportar las distintas operaciones que el componente implementa, así como las que necesita utilizar de otros componentes durante su ejecución. Y los requisitos son los que determinan las necesidades del componente en cuanto a recursos, como por ejemplo cuales son las plataformas de ejecución que necesita para funcionar. (2)

1.1.3. Definición de plugin

Un plug-in o extensión, es una aplicación que se relaciona con otra o más aplicaciones, con el fin de aportarle una nueva función y generalmente bastante específica. La aplicación principal es la encargada de ejecutar la que le fue adicionada y además interactúa mediante la Interfaz Programada de la Aplicación (por sus siglas en inglés API). (3)

1.1.4. Proceso de Integración de Datos.

El proceso de integración de datos se puede definir, como la forma de combinar datos procedentes de diversas fuentes, permitiéndole al usuario obtenerlos ya combinados y tener una vista de todos ellos. (4) Este proceso tiene gran importancia, ya que su realización, trae beneficios como satisfacer las

necesidades expuestas por los clientes y evitar grandes cargas de trabajo manual frecuentemente difícil de mantener y de desplegar.

Para el proceso de integración de datos existen varias herramientas. Entre ellas están Oracle Data Integrator (ODI), es una plataforma de integración bastante completa que cubre los requisitos de integración de datos, Informatica PowerCenter, Talend Open Studio, entre otras. Como se plantea con anterioridad, en el departamento Almacenes de Datos es utilizada la herramienta PDI, la cual brinda la posibilidad, de cubrir una amplia gama de necesidades en la integración de datos.

1.2. Herramienta Pentaho Data Integration

A continuación se muestra en este epígrafe, un estudio realizado sobre la herramienta Pentaho Data Integration, necesario para el desarrollo de la investigación y la comprensión de la situación problemática.

1.2.1. Antecedentes de Pentaho Data Integration

El belga Matt Casters [2]en el 2001 por las necesidades identificadas en su desempeño como desarrollador de almacenes de datos necesitó de una herramienta que le permitiera realizar el proceso de integración de datos de una manera automatizada. El lenguaje de programación que utilizó para desarrollar la herramienta fue Java y la librería gráfica AWT[3], esta última fue sustituida por la librería SWT[4] en años posteriores.

A la herramienta se le fueron añadiendo funcionalidades como acceso a bases de datos, tratamiento de ficheros y otras funcionalidades que se tradujeron en nuevos componentes. En la versión 2.0 se incluyó un

[2] Creador de la herramienta Pentaho Data Integration.

[3] Constituye una librería de clases orientada a objeto, para cubrir recursos y servicios de bajo nivel.

[4] Es un conjunto de componentes para construir interfaces gráficas en Java.

sistema de plugin para permitir el desarrollo de conectores de Kettle[5] con otros sistemas (como SAP[6]) y en 2005 fue liberado el código y puesto a disposición de todos en Javaforge[7].

El proyecto fue creciendo con rapidez y la comunidad se involucró en su desarrollo con mucha actividad, hasta que el Ketlle se integró a la órbita de Pentaho, adoptando el nombre de Pentaho Data Integration, que lo incluye como herramienta de Extracción, Transformación y Carga de datos (ETL) en su suite de productos. (5)

1.2.2. Breve descripción de Pentaho Data Integration

El PDI es un motor de transformación, y desde sus inicios fue diseñado para satisfacer las necesidades de los usuarios en la integración de datos, con una mayor facilidad. La herramienta desde su creación ha cumplido con todas las expectativas, siendo de gran aceptación para la mayoría de los usuarios que han interactuado con ella, además de contar con una interfaz agradable e intuitiva como se puede apreciar en la Fig 1. Las soluciones de Pentaho están escritas en el lenguaje Java, esto trae consigo que sean bastante flexibles, capaces de cubrir una amplia gama de necesidades en determinadas empresas o entidades. (6)

[5]Nombre con que surgió la herramienta

[6]Es un sistema de contabilidad -meta-contabilidad, popularmente descrito como un 'Enterprise Resource System' (ERP)

[7] Es un portal dedicado a hospedar proyectos desarrollados con licencias libres y relacionadas con la plataforma Java.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA SOBRE LA INTEGRACIÓN DE NUEVOS COMPONENTES A LA HERRAMIENTA PENTAHO DATA INTEGRATION

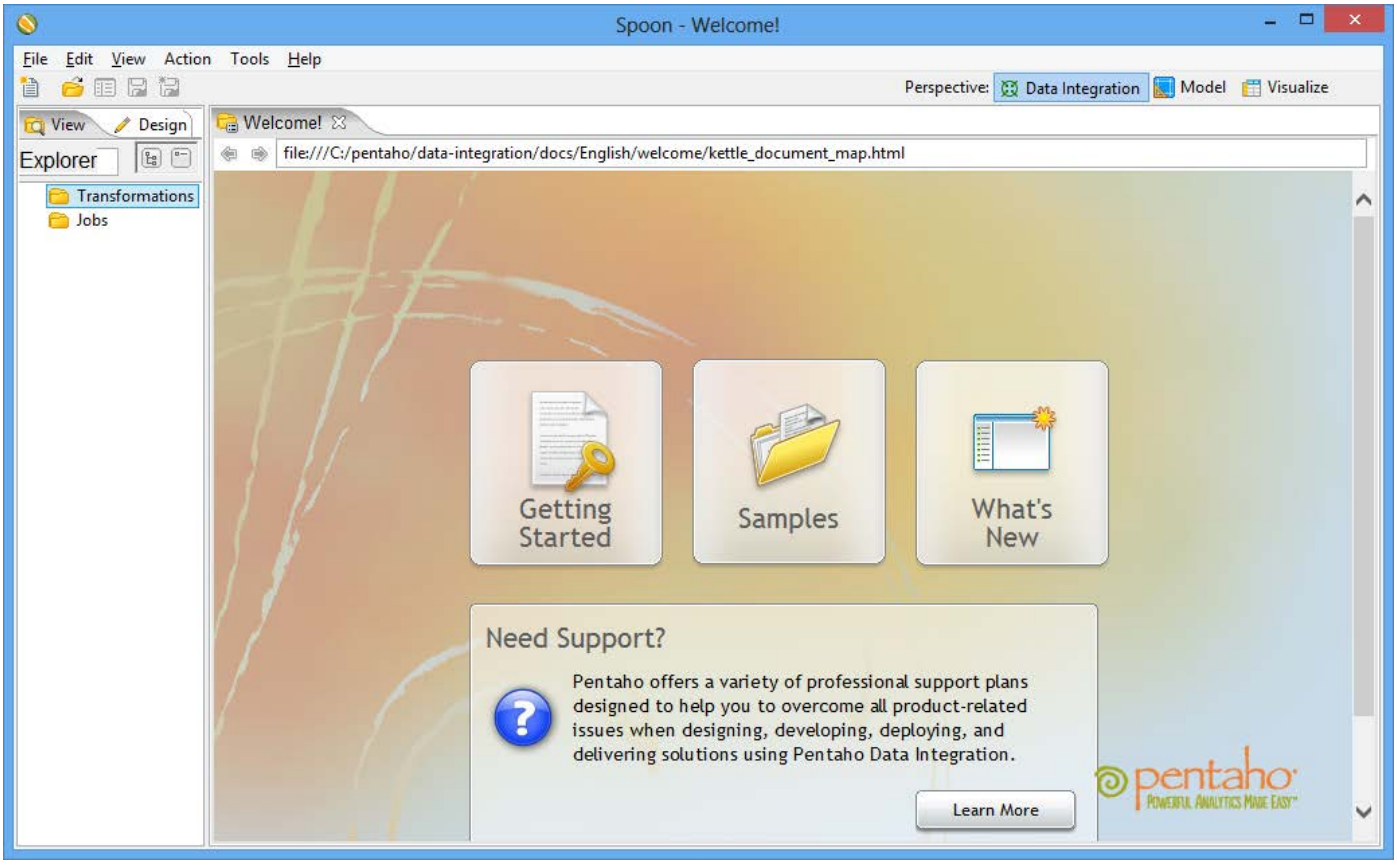


Fig. 1: Interfaz de Pentaho Data Integration

La herramienta permite implementar los procesos de extracción, transformación y carga de datos, la misma es de código abierto y está compuesta por cuatro herramientas fundamentales: SPOON para el diseño gráfico de las transformaciones, PAN utilizada para la ejecución de los trabajos y las transformaciones, CARTE para realizar el diseño de la carga de los datos y *KITCHEN* que se utiliza para la ejecución de los trabajos diseñados con CARTE.

1.2.3. Herramientas fundamentales de Pentaho Data Integration.

PDI está compuesto por cuatro herramientas principales, las cuales se describen a continuación: (5)

- **SPOON.** Es una aplicación de escritorio que utiliza una interfaz gráfica y un editor de transformaciones y trabajos. Proporciona la manera de crear puestos de trabajo de ETL complejas, sin tener la necesidad de leer o escribir el código.

- **PAN.** Es un proceso de línea de comandos independiente, que es utilizado para la ejecución de transformaciones creadas en el SPOON. El motor de la transformación de datos PAN, lee la información y escribe datos en diferentes fuentes de datos.

- **KITCHEN.** Es un proceso de línea de comandos independiente que se puede utilizar para ejecutar los trabajos. Es el encargado de ejecutar los diferentes trabajos que se diseñan en la interfaz gráfica de SPOON, ya sea en XML o en un repositorio de datos.

- **CARTE.** Es un contenedor web ligero, cuya función es configurar un servidor ETL dedicado a distancia. Esto proporciona la capacidad de ejecución remota, similares a las del Integration Server Data, pero no proporciona la programación, integración de seguridad y un sistema de gestión de contenidos.

1.2.4. Acciones que permite realizar el Pentaho Data Integration

Con la herramienta se pueden realizar una gran cantidad de tareas, que son de gran ayuda durante la integración de datos, entre estas tareas se destacan: (7)

- Brinda soporte para poder cambiar, enlazar dimensiones y realizar otras operaciones en el almacén de datos.
- Permite exportar de bases de datos a ficheros u otras bases de datos.
- Importar en bases de datos ficheros en formato Excel o texto.
- Migración de datos entre diferentes bases de datos.
- Explotación de los datos existentes en bases de datos (tablas, vistas, sinónimos,…)
- Enriquecer la información mediante búsqueda de datos en diferentes almacenes de información (bases de datos, ficheros de texto, hojas Excel,…).
- Limpieza de datos aplicando transformaciones de datos con condiciones complejas. Integración de aplicaciones.

En estas acciones son utilizados los componentes que contiene el PDI, mediante la aplicación de las transformaciones, varios de estos componentes han sido agregados en las diferentes versiones de la herramienta.

1.3. Desarrollo de nuevos componentes para PDI.

La herramienta PDI, a través de sus diversos componentes brinda un gran número de funcionalidades. En el transcurso de los años se han presentado situaciones, donde la herramienta no cumple con las necesidades de algunos especialistas. Debido a esto al PDI se le han integrado varios plugin, con el objetivo de cubrir todas estas nuevas necesidades que se presentan en el trabajo con la herramienta.

Para poder integrar un nuevo componente a la herramienta, debe contener cuatro clases fundamentales:

Data: la clase implementa la interfaz StepDataInterface. Es utilizada para el almacenamiento de datos únicos a un hilo de ejecución, cuando la transformación se ejecuta. Aquí es donde las conexiones de base de datos almacenan en caché los identificadores de archivos, y otras cosas necesarias durante la ejecución.

Meta: la clase implementa la interfaz StepMetaInterface. Es la encargada de entre otras funciones, inicializar los valores por defecto, proveer instancias de cada una de las clases del plugin, además de serializar los datos del componente como XML[8], lo que posibilita que una vez guardada la transformación, si se vuelve utilizar, no se pierdan.

Step: la clase implementa la interfaz StepInterface. Las instancias de Step hacen real el procesamiento de filas cuando se ejecuta la transformación. Cada hilo de la ejecución está representado por una instancia de esta clase. Además en ella se administra las instancias de las clases de datos y meta.

[8] Archivo donde se guardan los datos de la transformación.

Dialog: la clase de diálogo implementa la interfaz StepDialogInterface. Se muestra un cuadro de diálogo que le permite al usuario configurar el comportamiento del componente a su agrado. Esta clase de diálogo está estrechamente relacionada con la clase meta, que realiza un seguimiento de los ajustes seleccionados.

Además de estas clases fundamentales, los plugin tienen otras características que garantizan la unión con la herramienta PDI, accesibles a través de la interfaz del usuario. Existe un fichero (plugin.XML) que es el encargado de guardar toda la información externa necesaria para su visualización.

Descripción del archivo XML.

El archivo plugin.XML es sencillo de configurar. Su función principal es realizar la descripción y localización del plugin y decir a la herramienta cuál es la clase encargada de iniciar todo el proceso, cuya función es realizada por la clase Meta. El archivo cuenta con un identificador, este debe ser globalmente único, y no se debe cambiar. El ícono del archivo es una imagen con extensión png, esta es la que representará el plugin dentro del PDI. La descripción es el nombre del paso como aparece en el menú del árbol y la categoría es el nombre de la carpeta del árbol en la que aparece el plugin, o sea la clasificación que va a tener el mismo.

En la siguiente figura: Fig. 3 se muestra un ejemplo de la configuración de un archivo plugins.XML.

```
- <plugin id="DummyPlugin" iconfile="DPL.png" description="Dummy Plugin" tooltip="This is a dummy plugin test step" category="Transform"
  classname="be.ibridge.kettle.dummy.DummyPluginMeta">
  - <libraries>
      <library name="dummy.jar"/>
    </libraries>
  - <localized_category>
      <category locale="en_US">Transform</category>
      <category locale="nl_NL">Transform</category>
      <category locale="fr_FR">Transformation</category>
      <category locale="zh_CN">转换</category>
    </localized_category>
  - <localized_description>
      <description locale="en_US">Example plugin</description>
    </localized_description>
  - <localized_tooltip>
      <tooltip locale="en_US">This is an example for a plugin test step</tooltip>
    </localized_tooltip>
  </plugin>
```

Fig. 2: Ejemplo de un archivo plugin.XML

1.4. Metodología de desarrollo.

La metodología de desarrollo de software es parte esencial para la realización de cualquier proyecto o aplicación, es como una guía que define quién debe hacer, cómo y cuándo hacerlo.

Todo desarrollo de software es riesgoso y bastante difícil de controlar, de ahí la importancia de contar con una metodología que indique paso a paso las actividades que deben desarrollarse, las personas involucradas en la realización de las mismas, así como el rol que desempeñan estas en aras de lograr un

proceso de software exitoso. Las metodologías de desarrollo de software se pueden clasificar en dos grandes grupos:

Metodologías Tradicionales: metodologías orientadas al control de los procesos, estableciendo rigurosamente los roles, actividades, artefactos, herramientas y notaciones para el modelado. Requieren de una amplia documentación, muestran ciertas resistencias a los cambios debido a que, al consumarse uno, se ven afectados varios componentes del proceso de desarrollo del software y también se caracteriza por necesitar un proyecto de gran cantidad de participantes, pues requiere de un equipo de trabajo capaz de administrar un proceso complejo en varias etapas.

Metodologías Ágiles: metodologías orientadas a la interacción con el cliente y el desarrollo incremental del software**,** mostrando versiones parcialmente funcionales al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando, requiere de pequeños grupos de trabajo y son apropiadas para entornos volátiles. Son a diferencia de las tradicionales, más adaptables a los cambios. (8)

Para justificar y fundamentar qué tipo de metodología seleccionar, se realizó una comparación entre las metodologías ágiles y las tradicionales, la cual se muestra en la siguiente tabla (Tabla 2):

Tabla 2: Comparación entre las metodologías ágiles y las metodologías tradicionales. (9)

Metodologías ágiles	Metodologías tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparadas para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo de desarrollo).	Impuestas internamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.

No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (menos de 10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Muchos artefactos.
Pocos roles.	Muchos roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

A partir de lo analizado hasta el momento, entre los tipos de metodologías ágiles y tradicionales, se decidió utilizar una metodología ágil, debido a que sus características son bastantes similares a las del trabajo en realización, por ejemplo pocos integrantes en el equipo de desarrollo: en la investigación el equipo solo está conformado por un integrante, pocos roles: analista y desarrollador, la interacción entre el cliente y el equipo de desarrollo: en el presente trabajo se debe estar en constante comunicación con el cliente, mostrándole resultados parciales, entre otras características.

Dentro de las metodologías ágiles sobresalen SCRUM y XP (*Extreme Programming*). Además, en la UCI se emplea la metodología ágil SXP, desarrollada en el 2008 por UNICORNIOS: un grupo de proyectos de investigación y desarrollo que formó parte del polo productivo de Software Libre (SWL) de la UCI. La metodología SXP se basa en SCRUM para la gestión eficiente de los proyectos, y en XP, para la ingeniería de software.

Otra metodología ágil que sobresale es la OpenUp. Es un proceso, dirigido a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental apropiado para proyectos

pequeños y de bajos recursos; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo.

Para realizar una selección correcta de la metodología a utilizar como guía para el desarrollo del componente, se realizó un estudio de las metodologías ágiles que se mencionan anteriormente, mostrándose los resultados en la Tabla: 3.

Tabla 3: Tabla comparativa entre las principales metodologías ágiles

Metodología	Fortalezas	Debilidades
XP – SCRUM	El cliente es parte del equipo Evita las funcionalidades innecesarias. Puede ser aplicado en proyectos ya existentes. Revisiones frecuentes de lo que se está elaborando cada cierto período y lo que se presenta debe estar operativo.	Poca documentación Conceptos relacionados con la arquitectura y el diseño son difíciles de integrar. Solo proporciona soporte a la gestión de proyectos, cualquier otra disciplina queda fuera del ámbito
OpenUp	Detección de errores de manera oportuna debido a su desarrollo incremental. Cuenta con herramientas de ayuda. Define claramente sus roles, tareas y demás acciones en cuanto al desarrollo. Administra muy bien los requerimientos del Cliente.	Alargues de tiempo de desarrollo del proyecto. No admite proyectos grandes. Poca documentación

1.4.1. OpenUP

OpenUP mantiene las mismas características que RUP[9]: contiene el desarrollo iterativo, casos de uso y escenarios de conducción de desarrollo, gestión de riesgos y el enfoque centrado en la arquitectura. Tiene los componentes básicos que pueden servir de modelo a procesos específicos. Además, la mayoría de los elementos están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Es un proceso iterativo para el desarrollo de software que es mínimo porque solo incluye el contenido del proceso fundamental. Se caracteriza por ser un proceso completo ya que puede ser manifestado como proceso entero para construir un sistema. Igualmente, se comporta de forma extensible porque puede ser utilizado como base para agregar o para adaptar más procesos. (10)

El uso de esta metodología trae consigo beneficios como, por ejemplo, permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito, además permite detectar errores tempranos a través de un ciclo iterativo. OpenUP se rige por varios principios entre los que se encuentran colaborar para sincronizar intereses y compartir conocimiento, que no es más que promover buenas prácticas para el trabajo en equipos.

Los roles propuestos para la solución son los roles de analista y desarrollador. El analista es el encargado de recopilar los requisitos propuestos por el cliente y de entender el problema a resolver. Mientras que el desarrollador es el encargado de desarrollar el componente, realiza el diseño teniendo en cuenta la arquitectura utilizada, y realiza las pruebas unitarias y de integración.

1.4.2. Fases del OpenUp

Esta metodología, propone cuatro fases fundamentales como se muestra en la Fig 5: (11)

[9] Metodología tradicional de desarrollo de software.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA SOBRE LA INTEGRACIÓN DE NUEVOS COMPONENTES A LA HERRAMIENTA PENTAHO DATA INTEGRATION

1. Concepción

Concepción es la primera de las cuatro fases del ciclo de vida de OpenUP, en ella se realiza el entendimiento de los objetivos del proyecto, de esta manera se obtiene suficiente información para confirmar lo que el proyecto debe hacer. El objetivo de esta fase es capturar las necesidades de los clientes en los objetivos del ciclo de vida para el proyecto.

2. Elaboración

Es la segunda de las cuatro fases del ciclo de vida del OpenUP donde se tratan los riesgos significativos para la arquitectura. El propósito de esta fase es establecer la base para la elaboración de la arquitectura del sistema.

3. Construcción

Esta fase está enfocada al diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de esta fase es completar el desarrollo del sistema basándose en la arquitectura definida.

4. Transición

Es la última fase, cuyo propósito es asegurar que el sistema es entregado a los usuarios, y evalúa la funcionalidad y performance del último entregable de la fase de construcción.

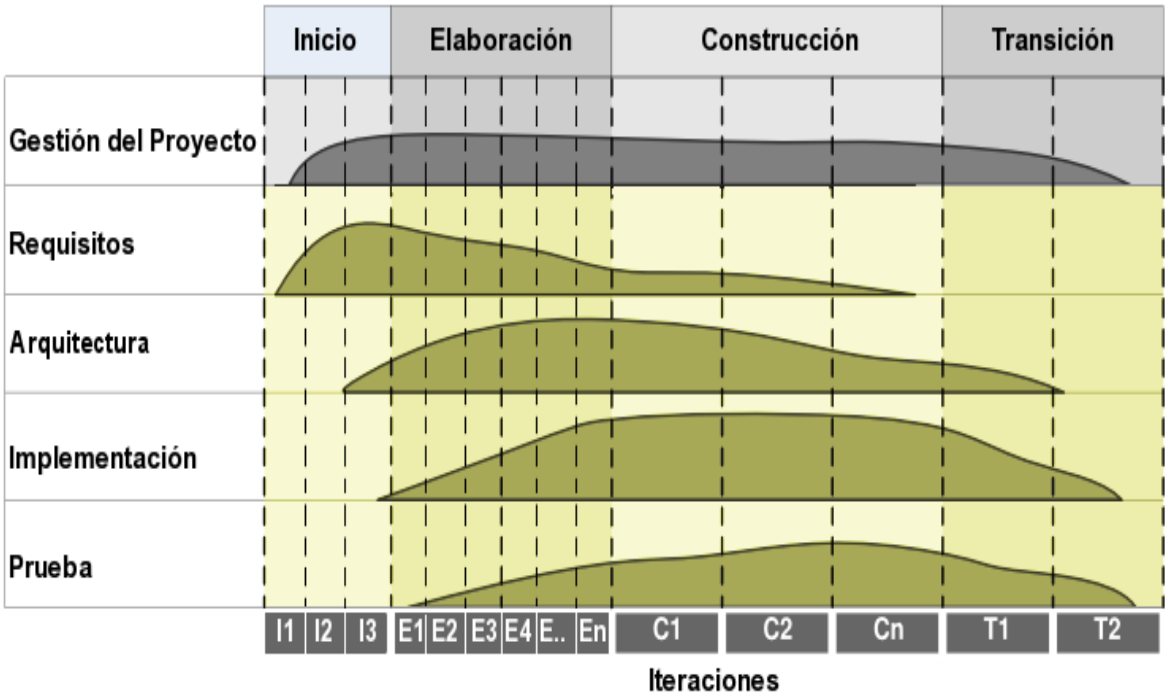


Fig. 3: Fases de OpenUp.

Para llevar a cabo la investigación, con apoyo de la metodología seleccionada anteriormente, se hizo necesario el uso de lenguajes, herramientas y tecnologías, para apoyar el desarrollo de la misma y así alcanzar los objetivos planteados.

1.4.3. Justificación de la metodología seleccionada

Por lo antes descrito se decide utilizar la metodología OpenUp, ya que esta es la que más se adapta a la investigación en curso, es utilizada en proyectos sencillos y con bajo presupuesto como el presente trabajo y además disminuye la probabilidad de fracaso del mismo, simplifica el desarrollo de software y disminuye en tiempo y costo los esfuerzos invertidos en el proyecto. Permite detectar errores en fases tempranas a través de un ciclo iterativo. Además es un proceso de desarrollo de software mínimamente suficiente, esto quiere decir que incluye solo el contenido fundamental. No provee orientación sobre temas en los que el proyecto tiene que lidiar, como son: el tamaño del equipo, el cumplimiento, seguridad, orientación tecnológica, entre otras. Además tiene un enfoque centrado al cliente y con iteraciones cortas.

1.5. Herramientas y tecnologías utilizadas.

1.5.1. Lenguaje de programación

Un lenguaje de programación se utiliza para expresar programas de ordenador. Está formado por un conjunto de símbolos, palabras claves utilizables y por reglas gramaticales para construir sentencias sintáctica y semánticamente correctas.

Java

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystem[10] a principios de los años 90. El lenguaje en sí mismo toma mucho de la sintaxis de Lenguaje de Programación C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. (12)Todos los usuarios que han interactuado con C++ encontrarán que Java es más sencillo, ya que se han eliminado ciertas características, como los punteros.

Ventajas de Java

Java tiene muchas ventajas, entre estas se tiene que es un lenguaje que se ejecuta en la mayoría de los sistemas operativos, inclusive en sistemas operativos móviles. Otra de las ventajas que presenta, es que es un software de distribución libre, ya que no es necesario pagar una licencia para poder comenzar a desarrollar en este lenguaje. Es un lenguaje bastante completo y poderoso, capaz de realizar una gran cantidad de tareas, mediante una librería y utilidades muy completas que facilitan la programación. (13)

[10] Empresa informática de Madrid, España.

Desventajas de Java

Java como cualquier otro lenguaje de programación presenta algunas desventajas, una de ellas es que puede presentar una ejecución lenta, debido al uso de la máquina virtual de Java, a diferencia de otros lenguajes de programación de más bajo nivel como lo es “C”, su velocidad de ejecución disminuye drásticamente. Otra de sus desventajas es que algunos usuarios lo consideran difícil de aprender, esto se debe a su compleja sintaxis, sin embargo su estructura es completa y organizada y semejante al lenguaje de C++, por lo que si se tiene conocimiento previo del mismo, Java podría ser fácil de aprender. (13)

1.5.2. Herramienta de modelado

Visual Paradigm 8.0

En la actualidad existen varias herramientas CASE, dentro de ellas destacan ArgoUML, Enterprise Architect, EasyCase, PowerDesigner y Visual Paradigm. Para el desarrollo de la investigación se decidió utilizar Visual Paradigm for UML 8.0, ya que presenta las siguientes ventajas y características: utiliza un lenguaje estándar, el cual es común para todos los integrantes del equipo de desarrollo, facilitando la comunicación entre los mismos. Por otra parte permite la realización de una gran diversidad de diagramas y la documentación de los mismos. La herramienta también proporciona abundantes tutoriales de UML[11] y demostraciones interactivas. (14)Además permite la integración con varias herramientas de lenguaje java, como son Netbeans y Eclipse (herramienta utilizada como entorno de desarrollo en la investigación) y tiene la ventaja de ser multiplataforma, capaz de exportar a XML, así como exportar los proyectos como imágenes. Se tuvo en cuenta para su selección, que es una herramienta capaz de generar de una forma rápida y eficiente los diferentes artefactos que propone la metodología seleccionada en la investigación y es bastante utilizada en el centro DATEC, así como en toda la universidad, durante el desarrollo de software.

[11] Lenguaje Unificado de Modelado.

1.5.3. Entorno de desarrollo integrado (IDE)

IDE Eclipse-Helios 2

Para el lenguaje Java son empleados en la actualidad diferentes entornos de desarrollos, cada uno con sus particularidades y características. Algunos de estos entornos de desarrollo son: JDeveloper, Netbeans, JavaStudio, Rational, BlueJ**,** Java Studio, Eclipse, entre otros. Para la investigación fue utilizado el IDE Eclipse en su versión Helios 2, del cual se realiza una breve descripción.

Eclipse es un poderoso IDE desarrollado inicialmente por IBM para el desarrollo de aplicaciones utilizando Java, actualmente su desarrollo es llevado a cabo por la Fundación Eclipse, una organización independiente y sin ánimos de lucro que fomenta una comunidad de código abierto, así como una serie de servicios. Eclipse se encuentra basado en plugin para brindar todas sus funcionalidades, a diferencia de otros entornos monolíticos que incluyen todas las funcionalidades, las necesite el usuario o no, y permite extenderse a otros lenguajes como C++ y PHP, agregándole el plugin correspondiente. (15)

Además se tuvo en cuenta que este IDE se utilizó durante el desarrollo del código fuente de la herramienta PDI y para la implementación de varios componentes, que han sido integrados a dicha herramienta, demostrando ser bastante eficiente para le realización de este tipo de proyectos.

1.6. Conclusiones parciales.

A partir de la revisión bibliográfica realizada sobre la herramienta PDI, profundizada en sus antecedentes, principales tareas y sus principales herramientas, permitió determinar cómo incorporar nuevos componentes a la misma. Además se analizaron las principales metodologías de desarrollo de software que existen en la actualidad y se determinó utilizar como guía para la investigación, la metodología ágil OpenUP teniendo en cuenta las características del sistema, lo que permitió establecer una guía, con el fin de realizar el proceso de desarrollo de software. Se determinó que herramientas eran las más eficientes para realizar la investigación, seleccionándose como IDE el Eclipse Helios, como herramienta CASE Visual Paradigm 8.0, lo que permitirá realizar los procesos de modelado e implementación del componente.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL COMPONENTE QUE PERMITA AÑADIR CONSTANTES DESDE CAMPOS ALMACENADOS EN BASE DE DATOS.

En este capítulo se realiza el análisis y diseño de la solución. En él quedan definidas todas las funcionalidades y características que debe cumplir el componente a través de los requisitos funcionales y requisitos no funcionales. Se identifican los casos de usos del sistema, realizando la descripción de cada uno de ellos, agrupando dentro de estos, los diferentes requisitos funcionales. Se modela el diagrama de clases del diseño, abordando acerca de los patrones utilizados, patrones de diseño y patrones arquitectónicos.

2.1. Propuesta de solución.

En este trabajo se desea desarrollar un componente que permita crear constantes a partir de campos almacenados en base de datos. El componente debe permitir al *usuario* conectarse a la *base de datos*, definiendo los parámetros de conexión. Una vez conectado a dicha base de datos, el usuario escoge la *tabla* que necesite en ese momento, después de seleccionar la tabla, selecciona el *campo* con el que va a trabajar y luego de ese campo seleccionado, obtiene el *valor* de la constante.

2.2. Descripción del modelo conceptual.

Un modelo conceptual es un artefacto de la disciplina de análisis, construido con las reglas de UML durante la fase de concepción, específicamente en la tarea construcción del modelo conceptual. Se encarga de capturar los objetos más relevantes en el contexto del sistema, representándolos en el entorno en el que se trabaja. (16)

En la presente investigación se realiza el modelo conceptual, el cual está conformado por varios conceptos fundamentales y se representan mediante un diagrama (Fig 6), lo que permite lograr una mayor comprensión del problema a resolver. A continuación se expone el diagrama que representa el modelo conceptual de la investigación:

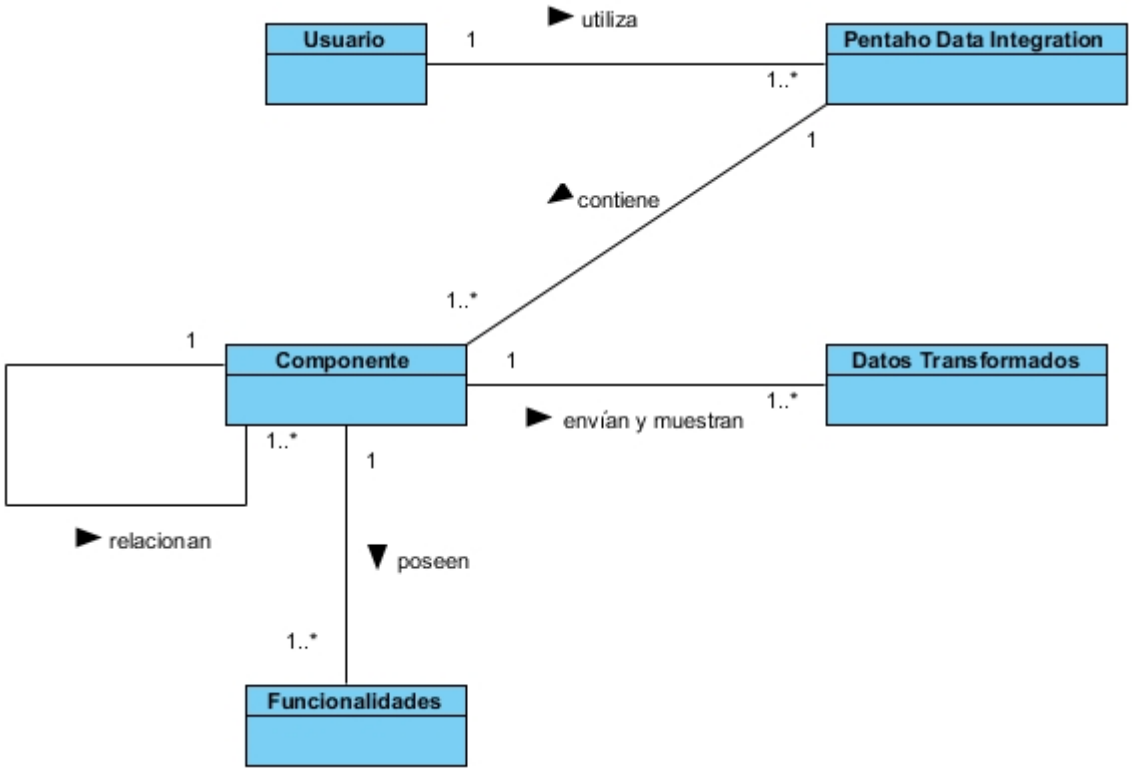


Fig. 4: Modelo conceptual

- **Usuario:** Es la persona que utiliza la herramienta PDI para la integración de datos.
- **Pentaho Data Integration:** Es una herramienta de integración de datos, donde se realizan las diferentes transformaciones a los datos, con el uso de diferentes componentes.
- **Componente:** Son una serie de funcionalidades necesarias para realizar el proceso de ETL.
- **Datos transformados:** Son los datos que pasan por el componente, a los cuales se les realiza una, varias transformaciones.
- **Funcionalidades:** Son las encargadas de asegurar que los componentes funcionen de forma correcta, para así asegurar que se alcancen buenos resultados.

2.3. Especificación de requisitos

La especificación de requisitos es un proceso fundamental para el desarrollo de cualquier software. Este tiene como principales objetivos permitir a los clientes describir con claridad lo que desean obtener, una vez terminado el software, por lo que deben tener activa participación durante este proceso. Por otra parte, permite a los integrantes del equipo de desarrollo construir un software que satisfaga las necesidades del cliente.

Una buena especificación de requisitos de software ofrece una serie de ventajas entre las que destacan el contrato entre cliente y desarrolladores, reduce el esfuerzo en el desarrollo, brinda una base para la estimación de costes y planificación, además de ser un punto de referencia para procesos de verificación y validación, y sirve como base para la identificación de posibles mejoras en los procesos analizados. (17)

2.3.1. Requisitos funcionales

Los requisitos funcionales son los encargados de describir las determinadas funciones o condiciones que debe brindar el sistema, en este caso, el componente. Describen además las posibles entradas y salidas, así como la respuesta que tiene el sistema ante algunas situaciones que se presenten en su interacción con el cliente.

Para la realización de esta investigación se identificaron los siguientes requisitos funcionales:

RF1: Conexión a la base de datos.

Descripción: Se realiza la conexión a la base de datos determinada.

Entrada: Parámetros de la conexión: Nombre de la conexión, el host, nombre de la base de datos, el usuario y la contraseña.

Salida: Conexión a la base datos definida y establecida.

RF2: Obtener tabla de la base de datos.

Descripción: Se lleva a cabo la selección de una tabla determinada en la base de datos.

Entrada: Conexión a la base de datos.

Salida: La tabla, con todos sus campos correspondientes.

RF3: Obtener campo de una tabla de la base de datos.

Descripción: Se lleva a cabo la selección del campo necesario, de una de las tablas de la base datos.

Entrada: Tabla de la base de datos.

Salida: El campo seleccionado de la tabla.

RF4: Obtener el valor del campo**,** de una tabla de la base de datos.

Descripción: Se obtiene el valor correspondiente al campo seleccionado.

Entrada: Campo de tabla.

Salida: Valor del campo seleccionado en la tabla de la base de datos.

RF5: Añadir el valor del campo al flujo de datos.

Descripción: Se añade el valor de la constante, al flujo de datos.

Entrada: Campo de tabla.

Salida: Valor de la constante añadido al flujo de datos.

2.3.2. Requisitos no funcionales.

Son aquellos requisitos que no se refieren directamente a las funciones específicas que debe entregar el sistema, sino a las características de éste, como por ejemplo: la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento. Los requisitos no funcionales surgen de acuerdo a la necesidad del usuario, por lo que es muy importante interactuar con él, con el fin de definir bien lo que desea.

Los requisitos no funcionales han de especificarse cuantitativamente, siempre que sea posible para que se pueda verificar su cumplimiento. Teniendo en cuenta todo lo anterior, se definieron varios requisitos no funcionales:

Requisitos de usabilidad:

- Para interactuar con el componente, los usuarios deben haber trabajado antes con el PDI y tener algún conocimiento del uso de sus componentes.
- Fácil de usar por parte de los usuarios. Debe tener una interfaz sencilla y atractiva, que permita crear interés al usuario y que brinde toda la información necesaria para interactuar con el componente, además de contar con mensajes claros asociados al contenido del componente.

Requisitos de hardware:

- Procesador: Celeron, 2.0GHz u otra superior.
- Espacio mínimo en disco duro: 250Mb
- Memoria RAM: 128Mb o superior.

Requisitos de software:

- Herramienta PDI, en una versión mayor a la 4.0.
- Máquina virtual de java en su versión 1.5 o una versión superior.

Requisitos de portabilidad:

- Después de haberse integrado el componente a la herramienta PDI, se podrá utilizar en diversos sistemas operativos como Windows7, WindowsXP, Linux MInt, Nova, ya que el PDI tiene como ventaja, ser una herramienta multiplataforma.

Requisitos de diseño e implementación:

- Se utilizan las herramientas Visual Paradigm for UML en su versión 8.0 como herramienta CASE y como IDE de desarrollo, el Eclipse en su versión Helios 2.
- Se utilizará el lenguaje java, para la implementación, ya que la herramienta PDI tiene un ambiente de implementación basado en este lenguaje.

2.4. Modelo de casos de uso del sistema.

El modelo de casos de uso del sistema representa las relaciones existentes entre actores y casos de uso. Los actores son terceros fuera del sistema que interactúan con él, puede ser cualquier persona, individuo, grupo, entidad, organización, máquina o sistema de información externo; con los que el sistema interactúa y los casos de uso son fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. (18)

Durante la investigación fue identificado un único actor del sistema, que se describe a continuación, en la Tabla 4:

Tabla 4: Descripción de los actores del sistema

Actor	Descripción
Usuario	Es la persona encargada de, a través del propio componente, conectarse a la base de datos, seleccionar la tabla necesaria, de la tabla seleccionar el campo específico, para luego obtener el valor de la constante de ese campo y añadirlo al flujo de datos.

2.4.1. Diagrama de caso de uso del sistema

El diagrama de casos de uso representado en la Fig. 7, describe las acciones que el usuario realiza mediante la utilización del componente, así como las funciones que este brinda.

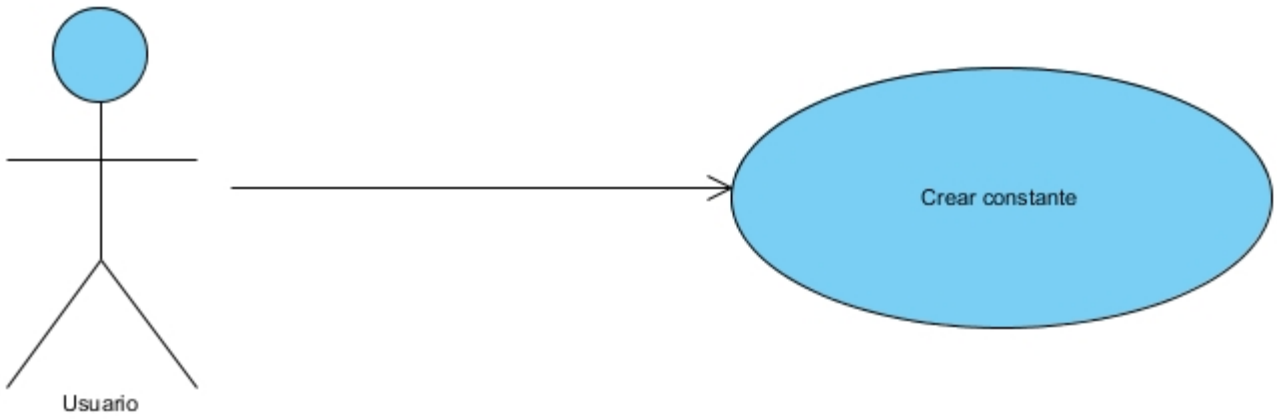


Fig. 5: Diagrama de caso de uso del sistema

2.4.2. Descripción textual de los casos de uso del sistema.

Luego de haber modelado el diagrama de casos de uso del sistema, se procede a describir textualmente el caso de uso: Crear constante.

Tabla 5: Descripción textual del caso de uso "Crear constante".

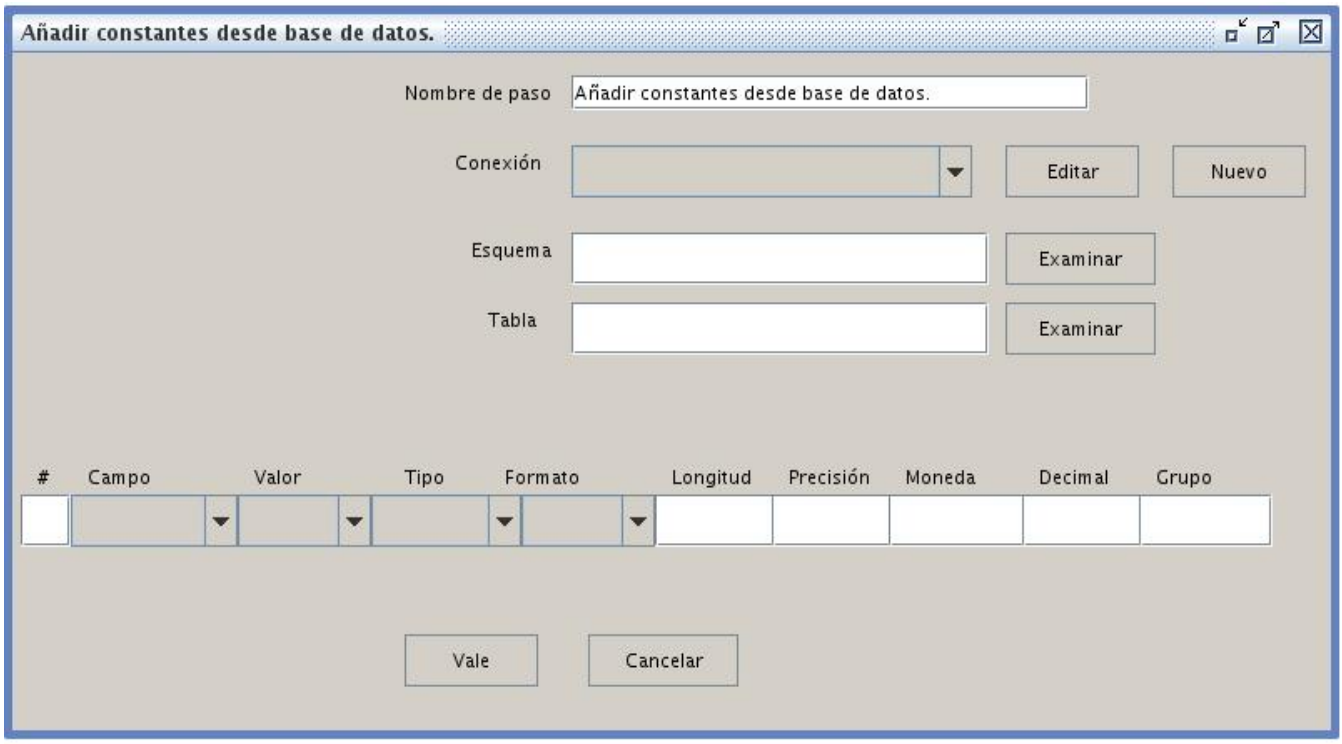
Caso de uso:	Crear constante.
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando es ejecutada la transformación, luego el sistema realiza la conexión a la base de datos, realizando las acciones necesarias. El caso de uso finaliza cuando son enviados los datos al próximo componente de la transformación.
Complejidad:	Media
Prioridad:	Alta
Precondiciones:	El actor debe abrir la interfaz del componente.
Pos-condiciones	Constante creada y añadida al flujo de datos.
Referencias:	RF1, RF2, RF3, RF4, RF5
Flujo Normal de Eventos	
Flujo Básico: Crear constantes.	
Acción del actor	**Respuesta del sistema**
1- Selecciona la opción *Nuevo,* para establecer una nueva conexión a la base de datos.	
	2- Muestra una interfaz, donde le permite al usuario definir los parámetros de la conexión: nombre de la conexión, el host, nombre de la base de datos, el usuario y la contraseña.

3- Presiona el botón *OK*.	
	4.1- Verifica que todos los campos de los parámetros estén definidos. 4.2- Verifica que los parámetros de la conexión son correctos, de estar bien definidos los parámetros, establece la conexión a la base de datos.
Flujos alternos al paso 4	
4.1- Parámetros de la conexión sin definir.	
	Muestra un mensaje solicitando definir los parámetros de la conexión que faltan.
4.2- Parámetros de la conexión incorrectos.	
	Muestra un mensaje de error, informando que hay parámetros incorrectos.
5- Selecciona la opción *Búsqueda de Esquema.*	
	6- Muestra una interfaz, donde le permite al usuario seleccionar el esquema que desee.
7- Selecciona el esquema.	
8- Selecciona la opción *Búsqueda de Tabla.*	
	9- Muestra una interfaz, donde le permite al usuario seleccionar la tabla que desee.

10- Selecciona la tabla.	
11- Selecciona la opción *Campo de tabla.*	
	12- Muestra un listado con todos los campos de la tabla seleccionada anteriormente.
13- Selecciona el campo de la tabla.	
14- Selecciona la opción *Valor.*	
	15- Muestra un listado con todos los valores del campo seleccionado anteriormente.
16- Selecciona el valor del campo.	
17- El usuario crea la constante, con el valor obtenido, definiendo el nombre, tipo, formato, longitud, precisión, moneda, decimal, grupo y campo de la tabla y luego presiona el botón *Vale.*	
18- Ejecuta la transformación.	
	19- Inicia la ejecución del componente.
	20- Realiza el procesamiento de los elementos de la clase *constanteStepMeta* y los elementos de la clase *constanteStepData*.
	21- Adiciona la constante al flujo de datos.
	22- Finaliza la ejecución del componente, y envía los datos obtenidos hacia el próximo presente en la transformación.

23- Verifica los resultados arrojados durante la ejecución del paso.	

Prototipo de interfaz



Flujos Alternos	
Acción del actor	**Respuesta del sistema**
Si hace clic en el botón “Cancel” no se crea la constante y se termina el caso de uso.	

2.5. Matriz de trazabilidad para los casos de usos.

La matriz de trazabilidad es la técnica mediante la cual se puede establecer una relación entre los diferentes elementos del desarrollo del software y los requisitos funcionales, posibilitando también determinar con qué caso de uso, se cubre cada uno de los requisitos funcionales, además de asegurar la ejecución de todos los elementos del desarrollo del componente.

Tabla 6: Matriz de trazabilidad para los casos de usos.

	RF1	RF2	RF3	RF4	RF5
CU1	X	X	X	X	X

CU: Casos de uso

RF: Requisitos funcionales.

2.6. Patrón arquitectónico.

Un patrón arquitectónico es de vital importancia ya que la estructura de un sistema influye directamente sobre la capacidad que presenta un sistema para satisfacer los atributos de calidad. Algunos de estos ejemplos de estos atributos de calidad son el tiempo de respuesta del sistema a las peticiones que se le hacen, la usabilidad, que tiene que ver con qué tan sencillo les resulta a los usuarios realizar determinadas operaciones con el sistema, como determinar qué tan sencillo es realizar cambios al mismo. (19)

En el desarrollo de la investigación se utilizó el patrón arquitectónico basado en plugin, como se observa en la Fig. 9. Según Johannes Mayer, este patrón se ve de forma independiente, ya que no posee ninguna variante arquitectónica. Garantiza el diseño de una aplicación con el objetivo de soportar varios plugin, permitiendo que la misma se extienda en tiempo de ejecución mediante la carga dinámica de módulos o clases que no conoce durante la compilación, por lo que el sistema puede alcanzar nuevas funcionalidades, sin afectar las funcionalidades ya existentes, lo que vuelve más eficiente a este propio sistema

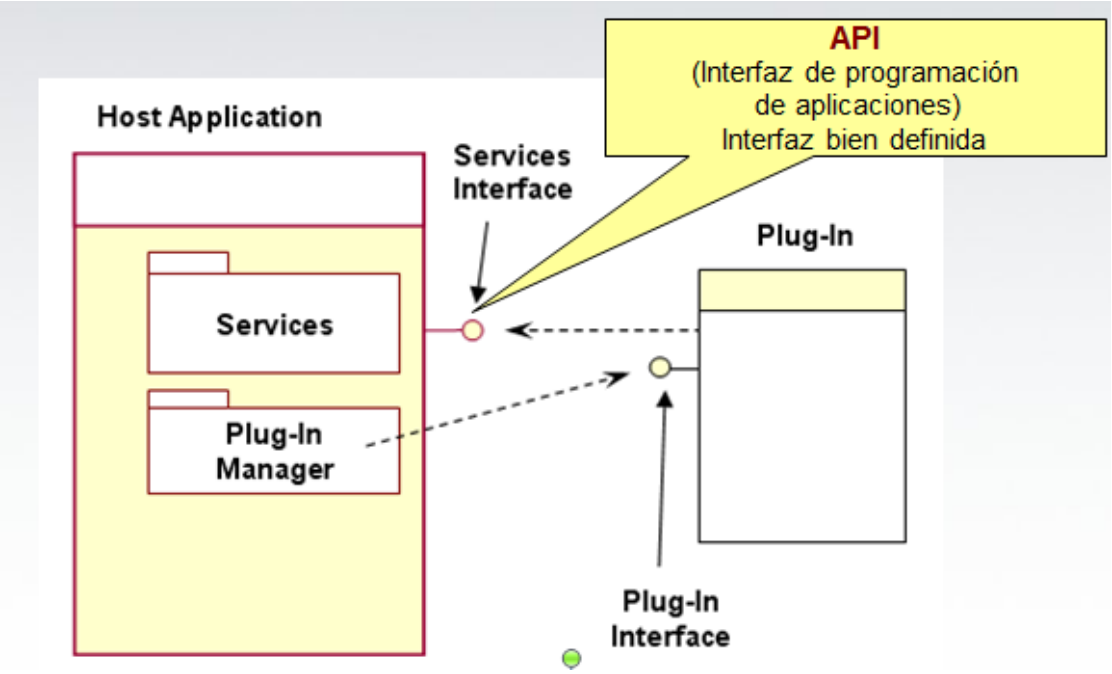


Fig. 6: Estilo arquitectónico basado en plugin.

2.7. Modelo del diseño del sistema.

El modelo del diseño describe la realización del caso de uso descrito anteriormente. Es considerado un elemento esencial en la realización de las actividades de ejecución y prueba y está basado en el análisis y los requisitos de la arquitectura del sistema. A través del modelo del diseño se definen las clases, subsistemas e interfaces, las relaciones entre ellas y las colaboraciones que llevan a cabo los casos de uso.

2.7.1. Diagrama de clases del diseño del sistema.

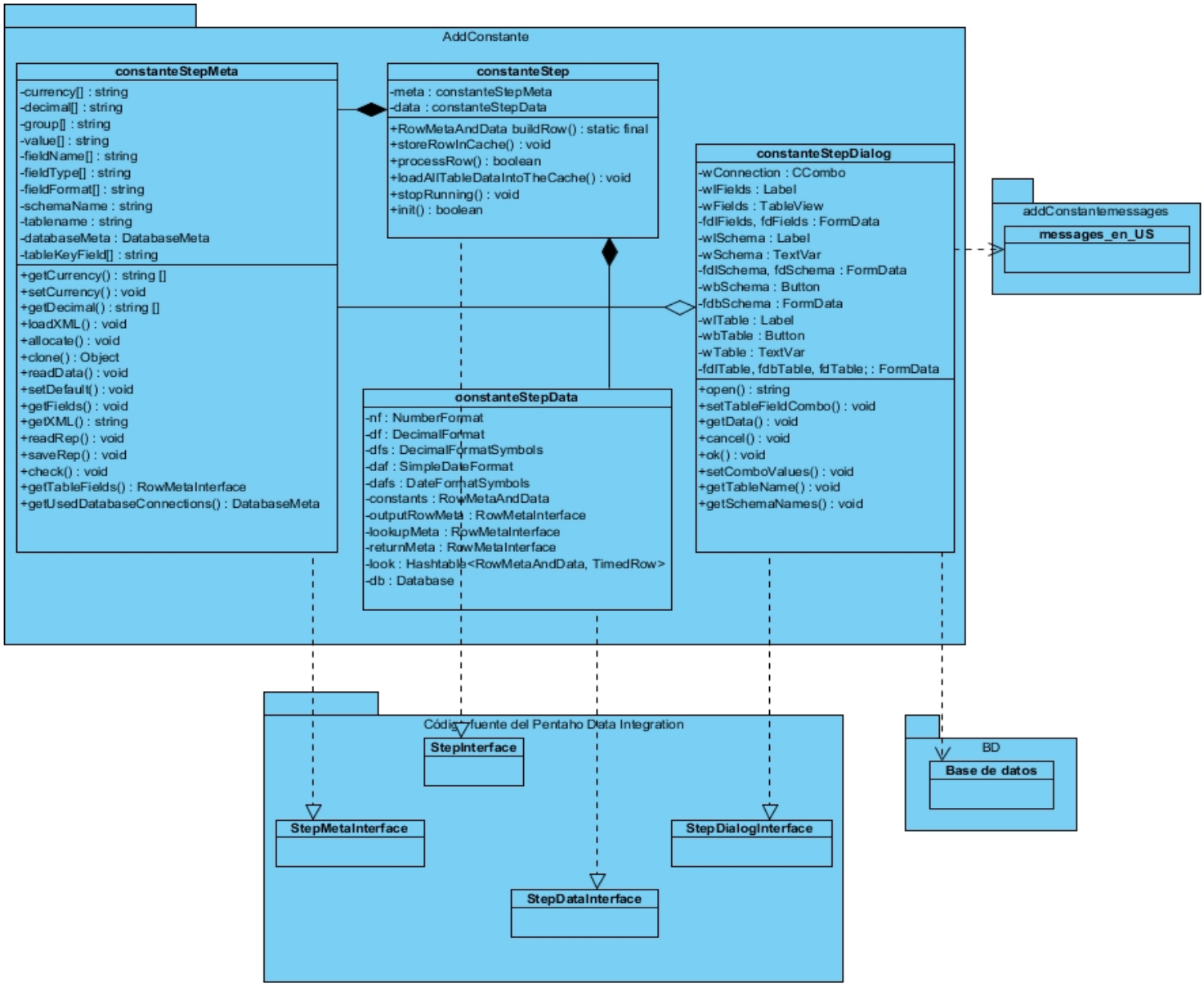


Fig. 7: Diagrama de clases del diseño

2.7.2. Descripción de las clases fundamentales del diseño.

constanteStep: esta clase es la encargada de implementar la interfaz StepInterface. Crea instancias de las clases constanteStepMeta y de la clase constanteStepData, con el fin de administrarlas y mediante sus

propias instancias realiza el procesamiento de filas, una vez ejecutada la transformación. Cada instancia de la clase representa un hilo de ejecución.

constanteStepMeta: esta clase es la encargada de implementar la interfaz StepMetaInterface. Provee instancias de las clases presentes en el componente (constanteStepData, constanteStepDialog**,** constanteStep). Contiene los métodos que permiten serializar los datos del componente como XML, e inicializa los valores de la constante por defecto.

constanteStepData: esta clase es la encargada de implementar la interfaz StepDataInterface. Una vez ejecutada la transformación, almacena los datos de cada hilo de ejecución, además de los datos de la conexión y de la constante creada.

constanteStepDialog: esta clase es la encargada de implementar la interfaz StepDialogInterface. Es la encargada de mostrar al usuario la interfaz visual del componente, permitiendo la configuración del mismo. Además contiene los métodos funcionales que permiten la conexión a la base de datos, seleccionar el esquema, entre otros métodos no menos importantes.

2.8. Patrones utilizados.

2.8.1. Patrones de diseño.

Los Patrones Generales de Asignación de Responsabilidad (GRASP), describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Son considerados como una serie de buenas prácticas para el diseño del software. (20)

Para la realización fueron necesarios utilizar varios de estos patrones de diseño, a continuación se mencionan los que fueron utilizados:

Creador:

Es el encargado de asignar la responsabilidad de que una clase X, cree un objeto de la clase Y, solo en el caso de que:

Y contiene a X

Y es una agregación (o composición) de X

Y almacena a X

Y tiene los datos de inicialización de X (datos que requiere su constructor)

Y usa a X

El uso de este patrón se ve evidenciado en el diagrama de clases Fig. 8, ya que la clase *constanteStep* crea instancias de la clase *constanteStepMeta* y de la clase *constanteStepData.*

Bajo acoplamiento:

Es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Debe haber pocas dependencias entre las clases. Este patrón se utilizó a la hora de realizar el modelado del sistema, asegurando que no exista demasiada dependencia entre las clases. Este patrón se evidencia en el sistema, ya que entre las clases existentes en el diagrama (Fig. 8), existe una clase controladora *constanteStep* que realiza toda la gestión de los objetos de la clase constanteStepMeta y de la clase constanteStepData, garantizando que si sufre algún cambio, afecte lo menos posible a las demás.

Experto:

Asignar una responsabilidad al experto en información, la clase que tiene la información necesaria para realizar la responsabilidad. Se evidencia cuando una clase contiene todos los datos o atributos necesarios, para realizar una labor específica. Este patrón se puede apreciar en más de una clase del sistema, por ejemplo en la Fig. 8, la clase *constanteStepDialog,* posee toda la información necesaria para realizar la interfaz visual del componente.

Controlador:

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Un ejemplo donde es posible apreciar el patrón controlador, es en la clase *constanteStep (*ver en la Fig. 8*)*, esta es la clase encargada de controlar todo el proceso, cuando se hace la ejecución del componente en el Pentaho Data Integration.

Alta cohesión:

Es una medida de la fuerza con la que se relacionan y del grado de focalización de las responsabilidades de un elemento, cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto. Este patrón se ve evidenciado en todas las clases del componente (Fig. 8), ya que cada una de ellas realiza una tarea específica para lograr un buen funcionamiento del mismo, por

ejemplo la clase *constanteStepData,* tiene como tarea almacenar los datos del componente, garantizando que haya alta cohesión.

2.9. Conclusiones parciales

Con la descripción del modelo conceptual y su elaboración, se logra una mayor comprensión de la solución, para así potenciar el desarrollo del componente para la herramienta PDI. Se definieron los requisitos funcionales y los no funcionales, lo que permitió conocer qué funcionalidades y características debía presentar el componente. Se realizó el diagrama de casos de usos del sistema, realizando la descripción del mismo y se agruparon en él todos los requisitos funcionales identificados, permitiendo solucionar problemas existentes. Se realizó el diagrama de clases del diseño, en el cual se identificaron los patrones utilizados, los de diseño y el patrón arquitectónico basado en plugin, el uso de estos patrones posibilitó una correcta asignación de responsabilidades a cada una de las clases.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA DEL COMPONENTE QUE PERMITA AÑADIR CONSTANTES DESDE CAMPOS ALMACENADOS EN BASE DE DATOS.

En este capítulo se hace referencia a todos los elementos necesarios utilizados durante la implementación del componente. Se realizó el diagrama de componentes del sistema, mostrando la relación que existe entre los diferentes elementos que componen el diagrama. Se definieron los estándares de codificación utilizados y además fueron realizadas las pruebas al componente para comprobar su funcionamiento.

3.1. Diagrama de componentes

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre sus elementos. Los mismos pueden ser simples archivos, paquetes o bibliotecas.

A continuación se muestra el diagrama de componentes para la solución propuesta y la descripción de los paquetes del subsistema.

El paquete **add_Constante**, contiene las clases del componente, son las encargadas de realizar todos los procesos para el funcionamiento del componente, por ejemplo el procesamiento de filas, el almacenamiento de datos, la conexión a la base de datos.

El paquete **Código Fuente del PDI**, contiene las clases de las cuales extienden las del propio componente. Estas clases se encuentran en el código fuente de la herramienta Pentaho Data Integration.

El paquete **addConstantemessages**, contiene los mensajes utilizados por la clase constanteStepDialog, los cuales son mostrados en la interfaz visual del componente.

El componente **Base de Datos**, representa la base de datos a la que se va a conectar el plugin, mediante la clase constanteStepDialog.

El componente **libraries,** son las bibliotecas que proporcionan funcionalidades y clases, que son necesarias para el funcionamiento del plugin.

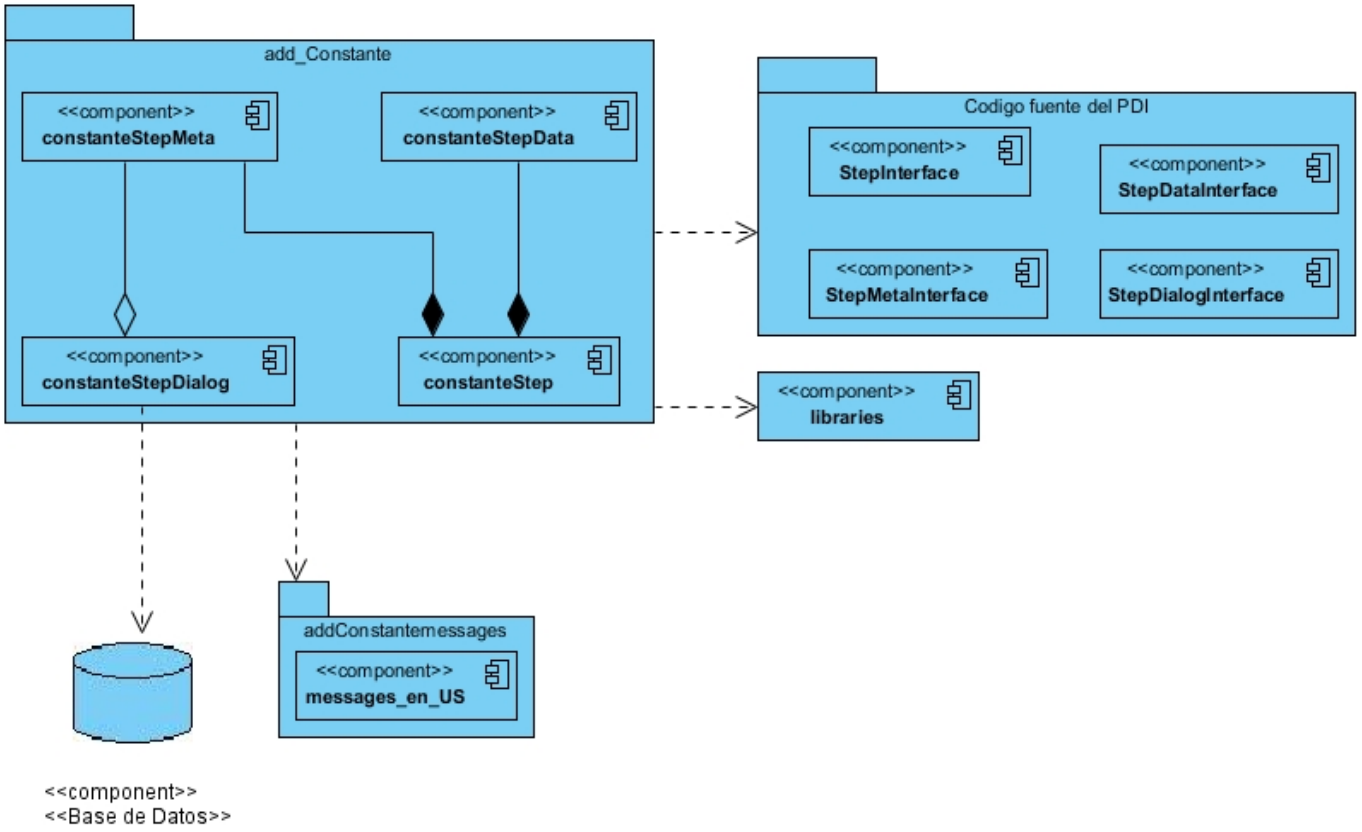


Fig. 8: Diagrama de componentes

3.2. Estándares de codificación.

Los estándares de codificación son por lo general reglas que se deben seguir para realizar la confección del código fuente. La principal ventaja que posee es que otros programadores puedan entender el código generado. Los estándares definen buenas prácticas de programación para lograr un código robusto, lo que ayuda en la calidad del software.

Reúnen un conjunto de técnicas de codificación sólidas y buenas prácticas de programación que propician que se genere un código de programación de alta calidad. Estas normas son de gran importancia para la calidad del producto de software. Algunas de las ventajas de utilizar estándares son:

- Facilitan el mantenimiento de una aplicación.

- Permiten que cualquier programador entienda y pueda mantener la aplicación.
- Mejoran la legibilidad del código, al mismo tiempo que permiten su compresión rápida.

Para la implementación del componente, fueron utilizados varios estándares de codificación, como se evidencia en la Fig 10 específicamente del lenguaje de programación Java.

3.2.1. Estándares de identación

Longitud de línea

Las líneas de códigos no exceden los 80 caracteres, ya que cuando esto ocurre, no son muy bien manejadas por algunas herramientas o IDEs de programación.

Rompiendo líneas

Cuando una determinada expresión, no entra dentro de alguna línea de código, se rompe de acuerdo a varios principios que plantea este estándar de codificación.

- Romper después de una coma.
- Romper antes de un operador.
- Preferir roturas de alto nivel, que de bajo nivel.
- Alinear la nueva línea con el comienzo de la expresión al mismo nivel de la línea anterior.
- Si las reglas anteriores llevan a código confuso o a código que se aglutina en el margen derecho, se debe romper justo 8 espacios en su lugar.

3.2.2. Estándares de formatos de los comentarios de implementación

Comentarios de bloque

Estos comentarios son utilizados para describir ficheros, métodos, estructuras de datos y algoritmos. Son usados al comienzo de cada método y también en el interior de estos, al mismo nivel que el código que describen.

Comentarios de fin de línea

Con el delimitador de comentario / / se convierte en comentario una línea completa o una parte de la línea.

3.2.3. Declaraciones

Cantidad por línea

Se realiza una declaración por línea ya que esto facilita los comentarios. Por ejemplo:

int nivel; // nivel de indentación

int tam; // tamaño de la tabla

Declaraciones de clases e interfaces

Para la declaración de las clases e interfaces se utilizaron las siguientes reglas de formato:

- Ningún espacio en blanco entre el nombre de un método y el paréntesis "(" que abre su lista de parámetros.
- La llave de apertura "{" aparece al final de la misma línea de la sentencia declaración.
- La llave de cierre "}" empieza una nueva línea indentada para ajustarse a su sentencia de apertura correspondiente, excepto cuando no existen sentencias entre ambas, que debe aparecer inmediatamente después de la de apertura "{".
- Las declaraciones de las clases comenzará con la palabra compuesta constante y en caso de que el nombre de la clase sea compuesto, las demás palabras comenzarán con letra mayúscula, evitando el uso de abreviaturas en las declaraciones.

Declaraciones de variables

- Las variables serán declaradas siempre con letra minúscula y no podrán comenzar con caracteres extraños. En caso de ser palabras compuestas, excepto la primera palabra, las demás comenzarán con letra mayúscula, evitando el uso variables largas. Las variables de un solo caracter solo se utilizaran en variables temporales, por ejemplo “int i = 0”, para la sentencia “for”.

```
public boolean processRow(StepMetaInterface smi, StepDataInterface sdi)
        throws KettleException {
    Object[] r = null;
    r = getRow();

    meta = (constanteStepMeta) smi;
    data = (constanteStepData) sdi;

    boolean sendToErrorRow = false;
    String errorMessage = null;

    // Get row from input rowset & set row busy!
    if (r == null) // no more rows to be expected from the previous step(s)
    {
        setOutputDone();
        return false;
    }

    if (data.firstRow) {
        // The output meta is the original input meta + the
        // additional constant fields.

        data.firstRow = false;
        data.outputMeta = getInputRowMeta().clone();
```

Fig. 9: Ejemplo de código donde se utilizan los estándares de codificación

3.3. Pruebas de software

Las pruebas de software es el instrumento utilizado para determinar el estado de la calidad de determinado software. En este proceso son ejecutadas una serie de pruebas ya sea a componentes del software o al propio software en su totalidad, para medir en qué grado el software cumple o no con los requerimientos del cliente. Son utilizados además casos de prueba, que se especifican de forma estructurada con el uso de las Técnicas de Prueba. Todo el proceso de pruebas, con los objetivos y los métodos y técnicas utilizados se plantean y describen en el plan de prueba. (21)

3.3.1. Niveles de pruebas utilizados

Las pruebas son realizadas en diferentes niveles de trabajo, algunos de estos son: de desarrollador, de sistema, de integración, de unidad y de aceptación. Durante la fase de prueba del componente fueron utilizadas pruebas a nivel de sistema, integración y de aceptación.

Nivel de sistema:

Estas pruebas son utilizadas para comprobar el comportamiento del sistema. Detectan fallas en el cubrimiento de los requisitos. Son realizadas por personas que no forman parte del equipo de desarrollo, con el objetivo de encontrar errores que no hayan sido detectados por los propios desarrolladores.

Nivel de integración:

Estas pruebas son utilizadas, con el objetivo de detectar errores que se presenten, asociados a la interacción del componente con otros ya presentes en el sistema. Además permiten verificar y detectar errores o interfaces incompletas de las clases.

Nivel de aceptación:

Estas pruebas son utilizadas para conocer la aceptación que tiene el software cuando es terminado e integrado a un sistema productivo, según las características y funciones especificadas por el cliente. Estas pruebas pueden ser realizadas a lo largo de semanas, descubriendo así errores acumulados que pueden ir degradando el sistema.

3.3.2. Tipos de pruebas realizadas al componente.

Al componente le fueron realizadas, una serie de pruebas para comprobar su funcionamiento. Las pruebas que se aplicaron son:

Pruebas de integración:

Durante la fase de prueba del componente, no fueron aplicadas ninguna de las técnicas de la prueba de integración, solo fueron utilizadas con el objetivo de detectar errores que se presenten, asociados a la interacción del componente con otros ya presentes en el sistema.

Las pruebas de integración fueron realizadas al componente, después de haber sido integrado a la herramienta PDI. Durante la realización de este tipo de pruebas, fue probado el componente en diversas

transformaciones (véase ejemplo en la Fig. 11), interactuando con un gran número de componentes, demostrando así su correcto funcionamiento, lo que posibilita su integración a la herramienta.

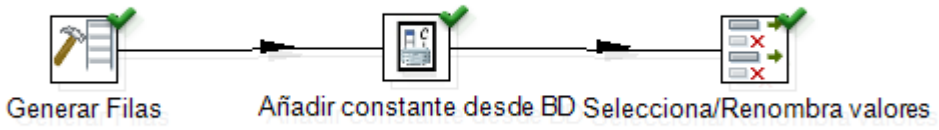


Fig. 10: Comprobando el funcionamiento del componente en una pequeña transformación

En la siguiente figura (Fig. 13) se evidencia el resultado satisfactorio de la transformación:

Execution Results

Execution History | Logging | Step Metrics | Performance Graph

#	Nombre paso	Numero Copia	Leído	Escrito	Entrada	Salida	Actualizado	Rejected	Errores	Activo
1	Añadir constante desde BD	0	10	10	0	0	0	0	0	Finalizado
2	Selecciona/Renombra valores	0	10	10	0	0	0	0	0	Finalizado
3	Generar Filas	0	0	10	0	0	0	0	0	Finalizado

Fig. 11: Resultado satisfactorio de la transformación

Pruebas de aceptación:

Las pruebas de aceptación son las encargadas de validar que el sistema da cumplimiento a los requisitos básicos de funcionamiento y a las expectativas del cliente, permitiendo a este determinar la aceptación del software, cuando es terminado e integrado a un sistema productivo. Es por ello que estas pruebas aunque son diseñadas por el propio equipo de desarrollo, deben ser realizadas por el usuario final, el cual plantea durante este periodo (puede ser a lo largo de semanas), todos los errores o deficiencias que encuentre, antes de dar la aprobación definitiva del sistema.

Se realizó este tipo de prueba al componente, para comprobar que haya cumplido con las expectativas de los especialistas de ETL del departamento Almacenes de Datos, de acuerdo a los requerimientos planteados. Una vez realizada las pruebas de aceptación por parte de los clientes y después de haber interactuado con el componente guiado por las pruebas diseñadas por el equipo de desarrollo, fueron

encontradas dos no conformidades. Las no conformidades fueron detectadas y corregidas, quedando satisfechos los especialistas con el resultado obtenido en la investigación.

Pruebas de funcionalidad:

- Función: Estas pruebas son las encargadas de validar las funciones, métodos, servicios y los casos de usos.
- Seguridad: Se asegura que el acceso a los datos o al sistema, sea efectuado por las personas autorizadas.
- Volumen: Verifica la habilidad que tienen los programas, para el manejo de grandes cantidades de datos, ya sea de entrada o de salida. (22)

De los tipos de prueba de funcionalidad, al componente se le aplicó el tipo función.

3.3.3. Métodos utilizados para realizar las pruebas de funcionalidad.

Las pruebas de caja negra: son realizadas sobre la interfaz del software y están centradas principalmente en los requisitos funcionales del software. Definen un conjunto de entradas con el objetivo de verificar la respuesta de los requisitos funcionales. (23)

Este método de caja negra presenta varias técnicas, entra las que se encuentran: análisis de valores límites, grafo de causa y efecto, partición o clases de equivalencia. Para la investigación fue seleccionada la técnica partición o clases de equivalencia. Esta técnica consiste en evaluar las clases de equivalencias[12], para un parámetro de entrada. Estas pruebas responden al nivel de prueba de Sistema.

3.4. Casos de prueba

Durante esta fase de las pruebas de software, se comprueba la implementación de cada una de las funcionalidades mediante el uso de casos de pruebas, utilizando la técnica seleccionada con anterioridad.

[12]clases de equivalencias: Son unos conjuntos de estados, los cuales son válidos o no válidos.

La descripción textual de los casos de usos, fueron la guía para diseñar cada uno de los casos de prueba. A continuación se muestra el caso de prueba realizado al caso de uso *Crear Constante.* Para este caso de uso se definen diez escenarios descritos en la Tabla 7.

Tabla 7: Descripción de las variables presentes en el caso de prueba para el CU *Crear constante*.

NO	Nombre del campo	Clasificación	Valor Nulo	Descripción
1.	Nombre	Campo de texto	No	Campo que permite una cadena de texto, donde se define el nombre de la constante.
2.	Tipo	Listado	No	Campo que permite seleccionar de un listado, el tipo de dato que va a tener la constante.
3.	Formato	Listado	No	Campo que permite seleccionar de un listado el formato de la constante.
4.	Longitud	Campo de texto	No	Campo que permite una cadena de texto, donde se define la longitud de la constante.
5.	Precisión	Campo de texto	No	Campo que permite una cadena de texto, donde se define la precisión de la constante.
6.	Moneda	Campo de texto	No	Campo que permite una cadena de texto, donde se define la moneda de la constante.
7.	Decimal	Campo de texto	No	Campo que permite una cadena de texto, donde se define el decimal de la constante.
8.	Grupo	Campo de texto	No	Campo que permite una cadena de texto, donde se define el grupo de la

				constante.
9.	Campo de tabla	Listado	No	Campo que permite una cadena de texto o seleccionar de un listado, el campo donde se encuentra el valor de la constante.
10.	Valor	Listado	No	Campo que permite una cadena de texto o seleccionar de un listado el valor de la constante.

En la siguiente Tabla 8 se definen tres variables, que son V que representa un valor válido, NV que representa un valor no válido e I que representa un valor irrelevante.

Tabla 8: Caso de prueba para el caso de uso *Crear constante.*

Escenarios de Crear constantes.	**Variables**			**Descripción de la funcionalidad**	**Respuesta del sistema**
	1	**2**	**3**		
Definir la constante correctamente.	V	V	V	En este escenario se define la constante que se va adicionar al flujo de datos de forma correcta.	Se introducen los datos de la constante y si son correctos, se adiciona la constante al flujo de datos.
Definir la constante con datos vacíos.	NV V V	V NV V	V V NV	En este escenario se define la constante que se va adicionar al flujo de datos con campos vacíos.	El sistema comprueba que los datos de la constante no están vacíos. Si los campos vacíos son obligatorios el sistema muestra un error.
Definir la constante con datos incorrectos.	NV V	V NV	I I	En este escenario se define la constante que se va adicionar al flujo de datos con campos	El sistema comprueba que los datos estén correctos. Si los datos son incorrectos se muestra un

				incorrectos.	mensaje de error.

3.5. Resultado de las pruebas

Después de haberse realizado las distintas pruebas a la aplicación, en una primera iteración fueron encontradas cuatro no conformidades, las cuales fueron detectadas y posteriormente solucionadas en otras dos iteraciones, como se muestra en la tabla siguiente (Tabla 9):

Tabla 9: Resultados de las pruebas realizadas a la aplicación.

Iteración	Fecha	Caso de prueba	Total de no conformidades	No conformidades resueltas	No conformidades pendientes
1era	25/05/2014	Caso de uso: *Crear constante*	4	4	0
2da	30/05/2014	Caso de uso: *Crear constante*	2	2	0
3ra	1/06/2014	Caso de uso: *Crear constante*	0	0	0

3.6. Resultado de la investigación

Después de haber realizado la implementación y haber realizado las pruebas al sistema, se obtiene un componente capaz de cumplir con todos los requisitos propuestos por parte de los clientes. A continuación se muestra el resultado de la investigación en la Fig. 12.

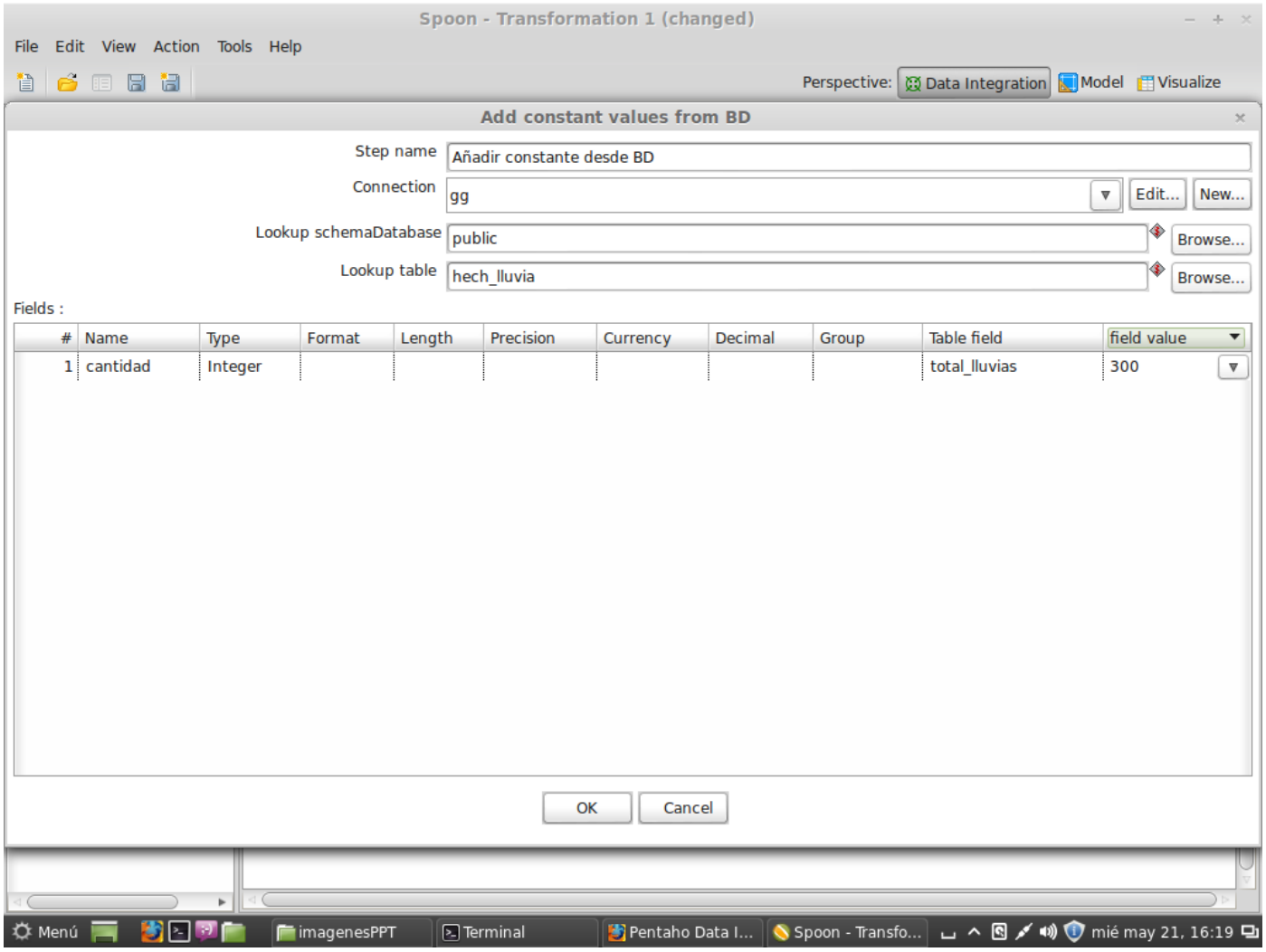


Fig. 12: Resultado de la investigación.

3.7. Conclusiones parciales

Con la realización del diagrama de componentes se representan los diferentes elementos que participan en el sistema, lo que permitió comprender la estructura y como se organizan los componentes que intervienen en la implementación. Fueron utilizados varios estándares de codificación, con el objetivo de

lograr un código claro para otros programadores y usuarios. Una vez terminada la implementación del componente le fueron realizadas una serie de pruebas, lo que permitió corregir las no conformidades y así garantizar el funcionamiento del componente.

CONCLUSIONES GENERALES

Con el desarrollo de la investigación se arribaron a las conclusiones siguientes:

- Con el estudio detallado sobre la herramienta de integración de datos PDI, se logró profundizar acerca de sus herramientas, acciones y funcionamiento, lo que permitió conocer como son integrados nuevos componentes a la herramienta, además de seleccionar la metodología y herramientas más eficientes para el desarrollo del componente.
- Con el análisis y diseño del componente de transformación, se logra una mayor comprensión de la solución, definiendo los requisitos, además de generar los artefactos que propone la metodología seleccionada, contribuyendo así al proceso de implementación del componente.
- Con la implementación del componente de transformación, se logra erradicar la deficiencia que generaba el componente *añadir constante*, permitiendo añadir constantes a partir de valores almacenados en base de datos.
- Con la realización de las pruebas al componente, fueron detectadas varias no conformidades, las cuales fueron corregidas, permitiendo satisfacer las necesidades expuestas por los clientes.

RECOMENDACIONES

- Se recomienda para próximas versiones del componente, que el mismo permita realizar varias conexiones a la misma base de datos.

REFERENCIAS BIBLIOGRÁFICAS

1. **Ecured.** *Ecured.* [En línea] 2000. [Citado el: 15 de diciembre de 2013.] http://www.ecured.cu/index.php/Transact_SQL.

2. **Scribd.** *Scribd.* [En línea] [Citado el: 14 de diciembre de 2013.] http://es.scribd.com/doc/73801129/Ingenieria-de-Software-Basada-en-Componentes.

3. **Valdespino Tamayo, Cecilia. Manzanillo.** *Arquitectura de Plug-in para Sistemas de Visualización Médica.*, Granma, Cuba : s.n., 2012.

4. **Oliva Alfonso, Debora.** *Propuesta de Herramientas para la Integración de Datos.* Instituto Superior Politécnico José Antonio Echeverria Cujae. La Habana : s.n., 2011. Tesis de Maestría.

5. **Sanchez Enriquez, Yusliel.** *Componente de Pentaho Data Integration para procesar cadenas de caracteres homogéneas.* Almacenes de Datos, Universidad de Ciencias Informáticas. La Habana : s.n., 2012.

6. **Ecured.** *Ecured.* [En línea] 2000. [Citado el: 20 de noviembre de 2013.] http://www.ecured.cu/index.php/Pentaho_Data_Integration.

7. **BI-Business Intelligence**-. *BI-Business Intelligence-.* [En línea] [Citado el: 22 de noviembre de 2013.] http://www.gravitar.biz/index.php/herramientas-bi/pentaho/caracteristicas-pentaho..

8. **Martinez Ajete, Mirisleydis y Francia Ofarril, Dayana.** *Desarrollo del Portal Web de la Organización Nacional de Bufetes Colectivos.* La Habana : s.n., 2014.

9. **Monografías.com.** *Monografías.com.* [En línea] 2010. [Citado el: 18 de enero de 2014.] http://www.monografias.com/trabajos67/metodologia-desarrollo-softwares/metodologia-desarrollo-softwares2.shtml..

10. **Brito Rodríguez, Julio César.** *Módulo Diseñador de Modelos para el Generador de Dinámico de Reportes v2.0.* Universidad de Ciencias Informáticas. La Habana : s.n., 2012. Trabajo de diploma.

11. **Ríos Salgado, Santiago, Hinojosa Raza, Cecilia y Delgado Rodríguez, Ramiro.** Aplicacion de la metodología OpenUp en el proceso de desarrollo del sistema de difusión de gestión del conocimiento de la ESPE. [En línea] septiebre de 2013. [Citado el: 10 de enero de 2014.] http://repositorio.espe.edu.ec/bitstream/21000/6316/1/AC-SISTEMAS-ESPE-047042.pdf.

12. **Atlantic International University.** Open Courses. [En línea] marzo de 2011. [Citado el: 2014 de enero de 12.] http://cursos.aiu.edu..

13. **Código Programación.** [En línea] abril de 2010. [Citado el: 30 de noviembre de 2013.] http://codigoprogramacion.com/cursos/java/47-introjava.html..

14. **Cabrera Gonzáles, Lianet y Pompa Torres, Enrique Roberto.** *Extensión de Visual Paradigm for UML para el Desarrollo Dirigido por Modelos de aplicaciones de gestión de información.* La Habana : s.n., 2011.

15. **Jiménez Milia, Joan Pablo y Ferrer Obregón, Roberto.** *Análisis de un IDE para múltiples plataformas con tecnologías y herramientas libres para desarrollar software educativo en formato multimedia. Subsistema de gestión de código.* La Habana : s.n., 2008.

16. ***El Proceso Unificado de Desarrollo de Software.*** España : s.n., 2007.

17. **Monferrer Agut, Raúl.** *Especificación de Requisitos Software según el estándar de IEEE 830.* s.l. : Universitat Jaume I, 2001.

18. **López Duque, Marleysi y Oceguera Ravelo, Raide.** *Herramienta en Matlab para la obtención de datos y ficheros electroencefalograma del proyecto Mapeo Cerebral Humano Cubano,.* Universidad de Ciencias Informáticas. La Habana : s.n., 2010. Trabajo de Diploma.

19. **Cervantes, Humberto.** *Arquitectura de Software.* 27, s.l. : SG, 2010.

20. **Larman, Craig.** *Uml y Patrones.* 2009.

21. **Pruebas de Software.** [En línea] mayo de 2005. [Citado el: 20 de abril de 2014.] http://www.pruebasdesoftware.com/laspruebasdesoftware.htm.

22. **Ecured.** [En línea] 2000. [Citado el: 20 de abril de 2014.] http://www.ecured.cu/index.php/Pruebas_de_Calidad_de_Software#Tipos_de_Pruebas_de_Software.

23. **Pressman, Roger.** *Ingeniería del software.* 2007.

BIBLIOGRAFÍA

1. **Booch, Grady, Jacobson, Ivar and Rumbaugh, James**. 2007. *El Lenguaje Unificado de Modelado.* ADDISON-WESLEY, 2007. 978-84-7829-087-1.
2. **Carrillos Perez, Isaias**. (2008). Metodología del desarrollo de software. (2008).
3. **Cecilia Valdespino Tamayo**. Arquitectura de Plug-in para Sistemas de Visualización Médica.
4. **Cervantes, Humberto.** *Arquitectura de Software.* 27, s.l. : SG, 2010.
5. **Craig Larman**. UML y Patrones. [En línea] www.publidisa.com/preview-libro-9788483229279.pdf
6. **Erich Gamma**. Design Patterns: Elements of Reusable Object Oriented Software [En línea] www.uml.org.cn/c++/pdf/DesignPatterns.pdf
7. **Groussard, Thierry**. 2012. Java, JAVA 7 - Los fundamentos del lenguaje. s.l. : ENI, 2012.
8. **Hernández, Cinolkis Cobas**. 2010.Desarrollo del Componente de Seguridad para el subsistema Mondrian perteneciente al sistema Pentaho. Universidad de las Ciencias Informáticas. La Habana : s.n., 2010. pág. 24.
9. **IAN Somerville**. Ingeniería del Software. [En línea]http://www.filecrop.com/ian-sommerville-ingenieria-de-software-septima-edicion.pdf.html
10. **Ian, Sommerville**; Ingeniería de software; 7ma edición; capítulo 26 pruebas del Software página 767.
11. **Ivar Jacobson, Grady Booch, James Rumbaugh,** El Procesos Unificado de Desarrollo de Software.
12. **Jacobson, Ivar, Booch, Grady and Rumbaugh, James**. 2000. El Proceso Unificado de Desarrollo. s.l. :Adison Wesley, 2000. ISBN 84-7829-036-2.

13. **Jiménez Milia, Joan Pablo y Ferrer Obregón, Roberto.** *Análisis de un IDE para múltiples plataformas con tecnologías y herramientas libres para desarrollar software educativo en formato multimedia. Subsistema de gestión de código.* La Habana : s.n., 2008.
14. **Johannes Mayer,** 2006. Ligthweight Plug-in Based Aplication Development. 2006.
15. **Kimball, Ralph y Caserta, Joe**. 2004. The Data Warehouse ETL Toolkit Practical: Techniques for Extracting, Cleaning, Conforming, and Delivering Data. [ed.] johnwiley&sons inc. ISBN 0764579231. Canada: Wiley Publishing, 2004.
16. **Kimball, Ralph. 2009**. Sub Sistemas de ETL, Billage. [En línea] 17 de 9 de 2009. http://bi.social.uoc.edu/smc/blog/34-subsistemas-etl-de-kimball. SBN: 0-764-57923-1.
17. **Martínez Ajete Mirileydis, Dayana Francia Ofarrill** "Desarrollo del Portal Web de la Organización Nacional de Bufetes Colectivos", Universidad de Ciencias Informáticas.
18. **Maurizio Lenzerini,** Data Integration: a Theoretical Perspective", Universita di Roma "La Sapienza". ACM PODS 2002.
19. **Monferrer Agut, Raúl.** *Especificación de Requisitos Software según el estándar de IEEE 830.* s.l. : Universitat Jaume I, 2001.
20. **Pan, D., Zhu, D., Johnson, K**. (2001). Requirements Engineering Techniques. Internal Report. Department of ComputerScience. University of Calgary. Canada.
21. **Pressman, Roger**; Ingeniería de software. Un enfoque práctico; 6ta edición. Capítulo 26 Gestión de la calidad.
22. **Pressman, Roger S**. Ingeniería del Software: Un enfoque práctico. Séptima Edición.
23. **Reynoso, Carlos**. 2005. Introducción a la Arquitectura de Software. Buenos Aires : s.n., 2005.
24. **Reynoso, Carlos. Kiccillof, Nicolas**. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Versión 1.0. [Citado el: 15 de diciembre de 2013].

25. **Sánchez Enriquez Yusliel** “Componente de Pentaho Data Integration para procesar cadenas de caracteres homogéneas”, Universidad de Ciencias Informáticas.

26. **Unidad de Compatibilización Integración y Desarrollo de Software para la Defensa (UCID)** (2012), Proceso de Desarrollo y Gestión de Proyectos de Software (Versión 1.5).

27. **Valdespino Tamayo, Cecilia.** *Arquitectura de Plug-in para Sistemas de Visualización Médica.* Manzanillo, Granma, Cuba : s.n., 2012.

ANEXOS

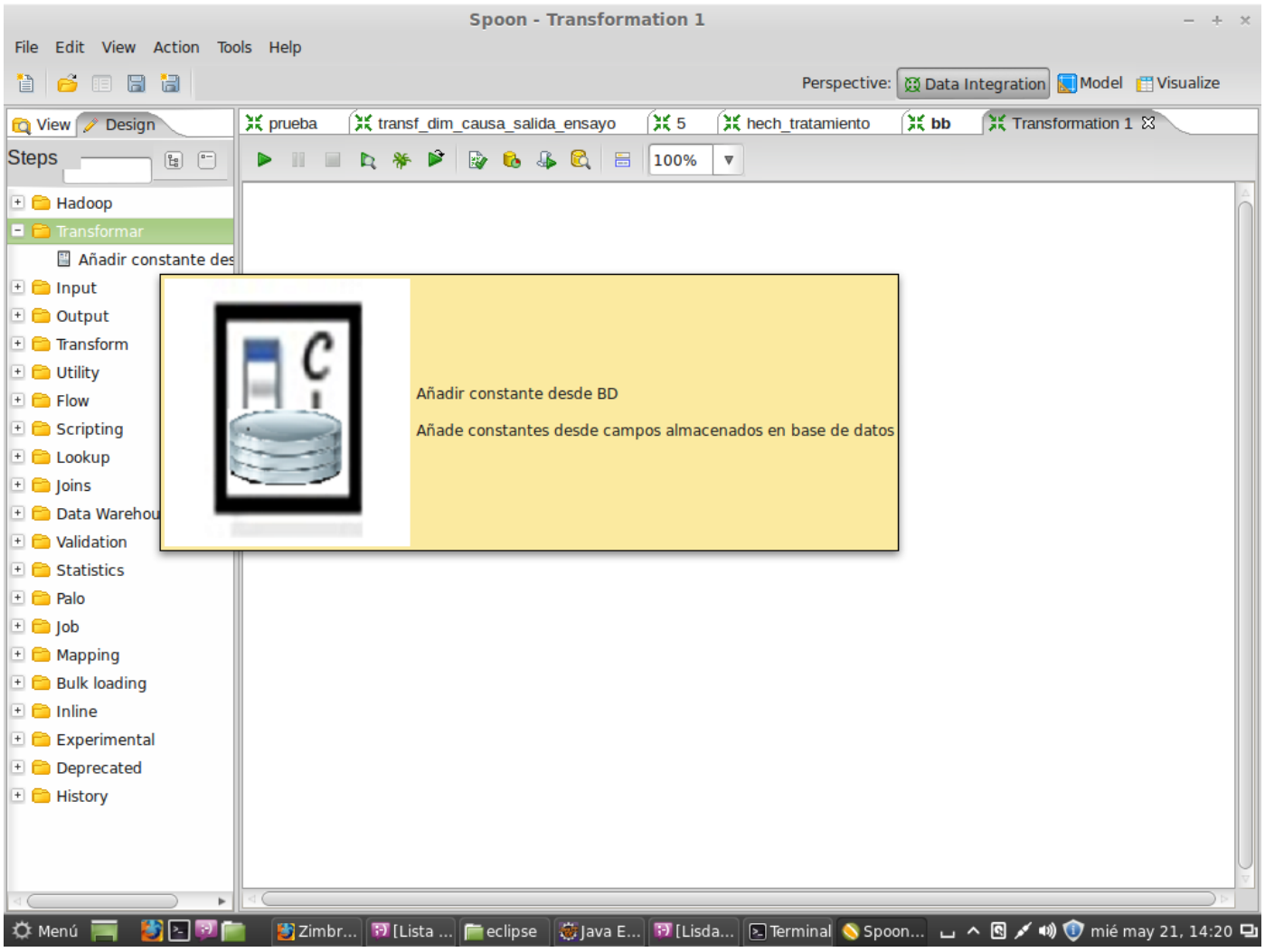


Fig. 13: Componente integrado a la herramienta PDI