Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Actualización de Herramientas y Aprendizaje en el Uso de Spoon PDI para la Gestión de Datos Geográficos

Autor: Javier Martínez Cuadrado

Tutor: Óscar Corcho García

Madrid, Junio 2022

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Actualización de Herramientas y Aprendizaje en el Uso de Spoon PDI para la Gestión de Datos Geográficos
Junio 2022

Autor: Javier Martínez Cuadrado

Tutor:
Óscar Corcho García
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

En el pasado, el OEG (Grupo de Ingeniería Ontológica) de la UPM empezó a trabajar con el IGN (Instituto Geográfico Nacional), con el objetivo de enlazar sus datos de tipo geográfico en la web, para así hacer la información más accesible, e incluso añadiendo nuevas formas de explotar dicha información. En uno de sus trabajos (Geographical Linked Data: a Spanish Use Case, 2010)[1] se exploró la posibilidad de enlazar ciertos datos a través de sus características geoespaciales, entre otras. Sin embargo, la idea se desarrolló exclusivamente para datos cuya geometría era sencilla, como puntos o curvas, y se dejó la alternativa de tratar con geometría más compleja (polígonos -> superficies) abierta para nuevos investigadores.

En primera instancia, se trató de seguir esta línea, pero la idea resultó ser muy ambiciosa y finalmente el proyecto fue enfocado al uso de la herramienta Spoon PDI y sus diferentes transformaciones.

[1] https://oa.upm.es/6167/1/Geographical_Linked_Data_A_Spanish_Use_Case.pdf

Abstract

In the past, the OEG (Ontology Engineering Group) from the UPM started working with the IGN (National Geographic Institute), aiming to link their geographic type data along the web to make information more accessible, and adding new techniques for a better exploit of that information. In one of their projects (Geographical Linked Data: a Spanish Use Case, 2010)[2] the possibility of linking certain data using their geospatial characteristics, among others, was explored. But the idea was developed exclusively for data with simple geometry, such as dots or lines, and the alternative of more complex geometry (polygons -> surfaces) was left apart, for new investigators to study.

At the beginning, this line was tried to be followed, but the idea turned out to be too ambitious and finally the project was focused on learning the bases of Spoon PDI and its transformations.

[2] https://oa.upm.es/6167/1/Geographical_Linked_Data_A_Spanish_Use_Case.pdf

Tabla de contenidos

1 Introducción

1.1 Objetivos iniciales

Los objetivos del proyecto serán; en primer lugar, generar los datos procedentes del IGN para actualizarlos y posteriormente convertirlos a un tipo de datos manejable que permita cumplir el segundo y principal objetivo: enlazar datos geográficos más complejos (desde un punto de vista geométrico) a través de su geometría (lagos, montañas, cordilleras, municipios, etc.). Para poder llevar a cabo los objetivos propuestos, habrá que asegurar que Pentaho funcione para la versión más actual del sistema.

Por lo tanto, para realizar el objetivo principal se seguirán los siguientes pasos:

- Revisar el plugin actual de Pentaho y comprobar que funcione para la última versión del sistema.
- Generar los datos del IGN y actualizarlos si fuese necesario.
- Convertir los datos a un formato más manejable para facilitar el 4º paso.
- Enlazar datos complejos a través de su disposición geométrica.

1.2 Contexto

En esta sección se tratarán las tecnologías actuales de representación de datos geográficos, así como las herramientas que se utilizarán durante el proyecto para llevar a cabo las tareas propuestas.

1.2.1 SIG: tipos

Los SIG (sistemas de información geográfica) son herramientas con las que se gestionan datos relacionados con el espacio en el que se encuentran (o lo que es lo mismo, su geografía). Al tratar entidades con ciertas características que además tienen un componente espacial se utilizan este tipo de sistemas. Por ejemplo: un río. De un río se puede sacar mucha información que no necesariamente tiene que ver con su posición geográfica. Su caudal, tipo de desembocadura (delta, simple, estuario o laguna), índice de contaminación, etc. Son tan solo algunos ejemplos que representan dicho tipo de información. Sin embargo, también se puede hablar de su forma (entendida como un conjunto de coordenadas que describen la manera en la que el río corre sobre la tierra), y por lo tanto tiene un componente geográfico; tiene una situación en el espacio. Es por ello por lo que un río puede ser una entidad manejada por un SIG. Otros ejemplos pueden ser municipios, comunidades autónomas, montañas, restaurantes, e incluso personas (entendidas como puntos que navegan sobre un mapa de coordenadas).

Los SIG tienen un gran conjunto de posibilidades en términos tanto de su uso como ámbitos. Desde la investigación de la historia en cierto lugar

(arqueología) hasta la toma de decisiones de mercado en base a la localización (marketing), pasando por un tema más relacionado con el presente trabajo: la relación de datos en la web (ingeniería informática, y más en concreto, linked data). Existen principalmente dos tipos de SIG: ráster y vectorial, y la decisión de utilizar uno u otro es situacional, ya que ambos presentan diferentes ventajas/desventajas.

1.2.1.1 Ráster

Se trata de un modelo que esencialmente busca la representación de datos en superficies continuas. El funcionamiento es el siguiente: se divide una porción de terreno en cuadrículas, y a cada cuadrícula se le asigna un valor que puede representar cualquier característica temática que se desee (puntos de mayor/menor carga en la planta de un edificio, zonas más/menos pobladas en un municipio…). Por supuesto, el número de cuadrículas puede aumentarse si interesa tener más resolución y por ende más precisión (a costa de un mayor coste computacional), o bien se puede disminuir si se busca más velocidad en el tratamiento de datos o un coste computacional menor [1].

1.2.1.2 Vectorial

El método antepone la precisión de los datos, haciendo que las superficies sean discretas. En el modelo vectorial, el almacenamiento de datos normalmente se divide en dos componentes: la primera componente almacena los datos espaciales, que se interpreta como un conjunto de puntos que pueden formar líneas, polilíneas, polígonos o incluso puntos sin más; y la segunda componente, que maneja datos temáticos de las distintas entidades.

La primera componente puede tener distintas estructuras de datos vectoriales que la soportan y el reparto de la información se puede incluso separar en varios ficheros. De menor a mayor complejidad, existen algunas como la lista de coordenadas, diccionario de vértices, ficheros DIME ("Dual Independent Map Encoding") o el arco/nodo [2].

1.2.1.3 Ráster vs vectorial

Por supuesto, un modelo no es mejor que el otro, como se ha dicho anteriormente. Simplemente ofrecen diferentes propiedades que se han de valorar en base al trabajo que se quiera realizar. A continuación, se muestran las principales ventajas y desventajas que ofrece cada modelo, con el fin de elegir uno de ellos después.

Ráster	**Vectorial**
Más compacto y rápido	Ocupa más memoria y lento
Más preciso en cuanto a la geometría	Más preciso en cuanto a la cantidad de información "temática"
2 dimensiones	3 dimensiones

Resolución “infinita”	Resolución limitada

Para poder elegir un modelo, recordemos que lo que se busca es asociar datos en función a la posición de las entidades en el mapa. Por lo tanto, lo que se busca es precisión geométrica; interesa conocer con exactitud la posición de las entidades en el mapa, más que sus características intrínsecas, como sería la altitud en cada punto de una cordillera o las distintas profundidades a lo largo de un río. En conclusión, el modelo adecuado para la tarea entre manos es el vectorial.

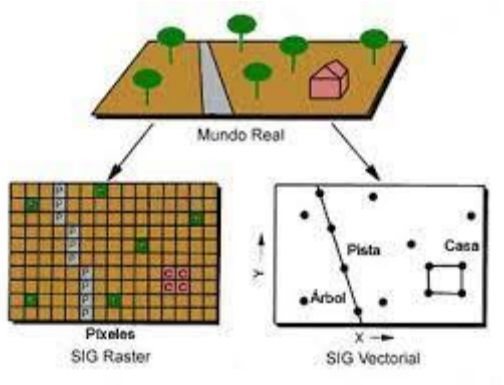

Figura 1: Representación gráfica de las diferencias entre los modelos ráster y vectorial.

1.2.2 Shapefile

Se trata de un modelo de representación de datos vectorial creado por ESRI (Enviromental Systems Research Insitute), y cuya peculiaridad reside en la repartición de la información en varios archivos. Cada uno de estos archivos contiene información específica referente a la entidad en cuestión. Después, la información de los archivos se relaciona a través de identificadores. Habitualmente hay 4 archivos diferentes (descritos a continuación), aunque puede haber más:

- .shp: contiene la información geométrica de manera vectorial. Puede haber puntos, líneas o polígonos.
- .shx: contiene un índice de las entidades geométricas del Shapefile.
- .dbf: contiene la información temática de las entidades: ya sean fechas, colores, clasificaciones…
- .prj: contiene información referente a la geometría: sistema de coordenadas empleado, unidades utilizadas, etc. Y sirve para referenciar a las entidades directamente sobre un mapa.

1.2.3 Spoon PDI

Pentaho Data Integration (PDI) es una herramienta desarrollada por la empresa japonesa Hitachi [3] que permite manejar datos mediante técnicas ETL

(extract, transform, load). Dichas técnicas incorporan 3 fases, tal y como indica su nombre: en la primera fase (extracción) se asegura que el trasvase de datos al sistema origen se realiza de forma correcta y sin que el sistema quede dañado; la segunda fase (transformación) implica la mayor carga cognitiva. En ella, el usuario debe conocer el objetivo que persigue, puesto que transformará los datos que haya en el sistema origen a su antojo; y finalmente la tercera y última fase (carga) consiste en transmitir los datos transformados al sistema destino.

Spoon no es más que la GUI (interfaz gráfica de usuario) de PDI. Se trata de una interfaz muy fácil de utilizar y que no exige a los usuarios una preparación extensiva, por lo que es muy amigable para ellos. El funcionamiento se basa en un "drag and drop", con el fin de formar una especie de puzle que en conjunto es capaz de tomar datos (en diversos formatos: ya sea .csv, .shp, etc.) y transformarlos según el criterio del usuario.

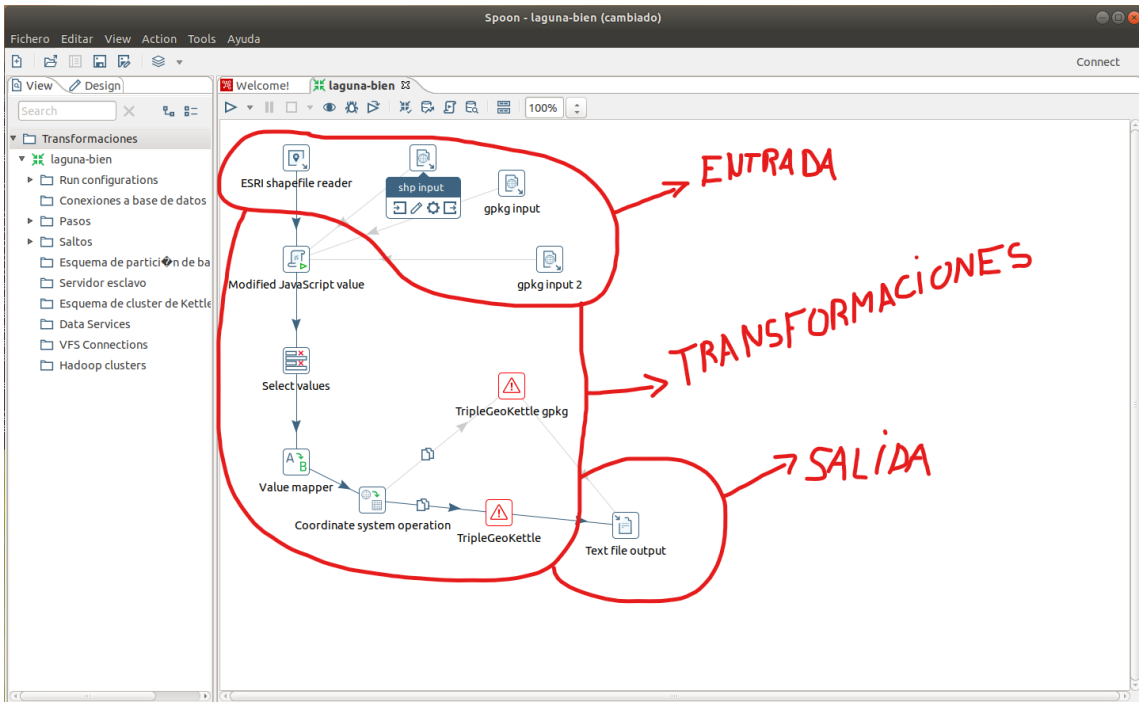

Figura 2: Pantallazo de Spoon PDI y distinción entre las diferentes fases.

Hay ciertos pasos que ya incluye PDI por defecto. Sin embargo, otros pasos no. Por eso existen los plugin, que añaden funcionalidades adicionales al software (en este caso PDI). En la imagen superior, dentro de la sección de "transformaciones" hay dos pasos que aparecen con un símbolo de peligro: se trata de TripleGeoKettle.gpkg y TripleGeoKettle. Los símbolos indican que es necesario o bien incluir un plugin para poder realizar esas transformaciones o bien no hacerlas (para eso habría que modificar el fichero .ktr que se pasó como argumento).

PDI dispone de un lector de Shapefiles. Se trata de ESRI shapefile reader (en la imagen superior, arriba a la izquierda). Este lector recibe únicamente dos de los archivos de los que se ha hablado en el punto 1.2.2: el .shp y el .dbf. Más adelante se verá que este lector no funciona por problemas desconocidos, y en su lugar se utilizará el lector de GIS, que solamente recibe el .shp.

1.2.4 IGN

El Instituto Geográfico Nacional (IGN) [4] fue creado en 1870 y hoy en día es una institución en la que recaen numerosas funciones, todas ellas, claro está, relacionadas con la geografía. El propósito de mencionar dicha institución en este trabajo no es otro que señalar sus intenciones de formar parte del movimiento "semantic-web". El IGN tiene a disposición de cualquier persona la información ligada con la institución, y una de las tareas que se plantean en el trabajo es recoger esa información y actualizarla. Así que, en este ámbito, el IGN no solo es una institución, sino que también es un portal de datos abierto. Concretamente será el portal del que se recojan los datos que se emplearán en el trabajo con el fin de enlazar datos a través de su geometría. Dichos datos se almacenan en formato Shapefile, que es de tipo vectorial, lo cual viene bien para cumplir el objetivo, tal y como se expone en el punto 1.2.1.3. En el caso del IGN, los Shapefiles vienen compuestos por los 4 archivos más usuales de los que se ha hablado en el punto 1.2.2. Sin embargo, solo se utilizarán el .shp y el .dbf.

1.2.5 Formato WKT (Well-known text)

Se trata de un lenguaje creado por el OGC cuyo objetivo es la estandarización de la representación de datos vectoriales en cadenas de texto. Con él se pueden representar puntos, líneas, polígonos, colecciones de elementos geométricos… El IGN usa este formato para el almacenamiento de sus datos geométricos. Un ejemplo se puede encontrar en el IGN[3].

Dependiendo de la geometría del objeto que se quiera representar, el formato será uno u otro. No merece la pena tratar cada formato posible para cada objeto en el presente trabajo debido a la extensión que supondría hacerlo. La información al completo queda disponible en la web si se quiere indagar [5]. Sin embargo, merece la pena mencionar el tipo POLYGON, puesto que será el tipo geométrico por excelencia a lo largo del trabajo. La cadena de texto para el caso de los polígonos es "POLYGON ((x1 y1, x2 y2, x3 y3… xn yn, x1 y1))", siendo "xi yi" las coordenadas de los puntos que describen el polígono. Nótese que el primer y el último punto del polígono han de ser el mismo. Esto es para formar una estructura cerrada (véase 2.4.2).

[3] https://datos.ign.es/recurso/btn100/laguna/3

2 Desarrollo

2.1 Capítulo 1: Preparación del entorno de desarrollo y generación de un pipeline desde cero.

Para empezar, se ha descargado PDI en su versión más actual (9) [6] y se han instalado los plugin necesarios para poder realizar la transformación .ktr. El primer plugin que se ha añadido ha sido "pentaho-gis-plugins" en su versión 9 [7]. Para ello, una vez descargado PDI y el plugin, hay que descomprimir el plugin y añadir la carpeta que contiene el .jar y la librería a la ruta "pdi-ce-9.2.0.0-290/data-integration/plugins/" que ya contiene por defecto un conjunto de plugins. Para el segundo plugin, se ha clonado el repositorio Github morph-geo[4] y se ha ejecutado instalador.sh (contenido en "morph-geo/TripleGeoKettle-PDI9/"), obteniendo el directorio "morph-geo/TripleGeoKettle-PDI9/target" que contiene tripleGeoKettle-oeg-1.jar. Ese .jar debe meterse en una carpeta, y esa carpeta será la que hay que añadir a "pdi-ce-9.2.0.0-290/data-integration/plugins/" como se hizo con el primer plugin.

Si los plugin se han añadido correctamente, al ejecutar spoon.sh y probar la transformación "morph-geo/transformaciones/laguna-tests.ktr", no se debería mostrar ningún fallo en las transformaciones tal y como se muestra en la imagen inferior (a diferencia de la figura 2).

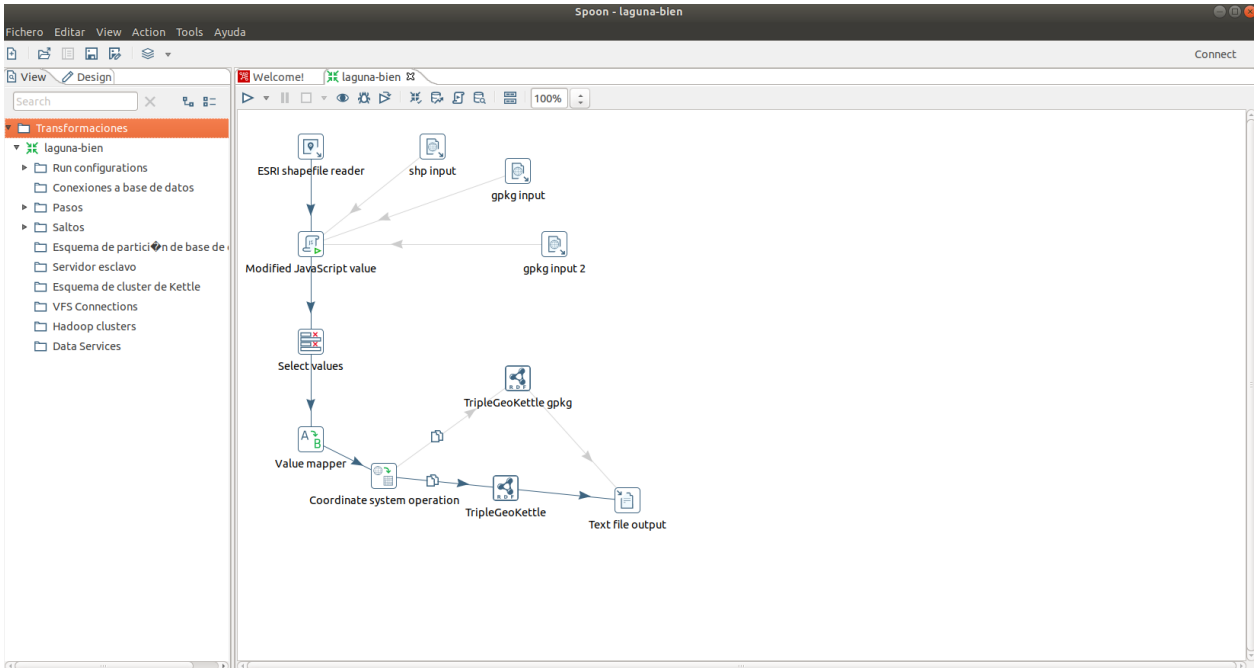

Figura 3: Pantallazo de Spoon PDI sin errores por dependencias con plugins.

[4] https://github.com/oeg-upm/morph-geo

El siguiente paso es debugear cada paso de la transformación. Si se ejecuta la transformación directamente, se verá que las rutas de los inputs y el output no se encuentran. Esto es porque las rutas especificadas por Beñat eran correctas para hacer la transformación en su máquina, pero no para nuestra máquina. Por lo tanto hay que modificarlas.

Para cambiar las rutas se ha de disponer en primer lugar de los ficheros .shp y .dbf de entrada. Estos ficheros se pueden encontrar por ahora en el repositorio del OEG[5]. Y se dice "por ahora" por un motivo; y es que este repositorio lleva sin ser actualizado 5 años. La intención actual es comprobar que PDI funciona (ajustar plugins y debugear). Más adelante se recurrirá al repositorio del IGN que contiene todos los datos actualizados para renovar el repositorio del OEG y realizar las transformaciones con esos datos. Volviendo al problema: en la ruta anteriormente especificada se encuentra el directorio comprimido "BTN100_0303S_LAGUNA.zip". Se descarga y descomprime. Dentro de él están los ficheros .shp y .dbf que hay que añadir a la ruta de "ESRI shapefile reader" y "shp input". Tal y como se puede apreciar en la imagen inferior (ejemplo para ESRI shapefile reader), se ha descomprimido la carpeta en el escritorio y se referencia a los nuevos ficheros.

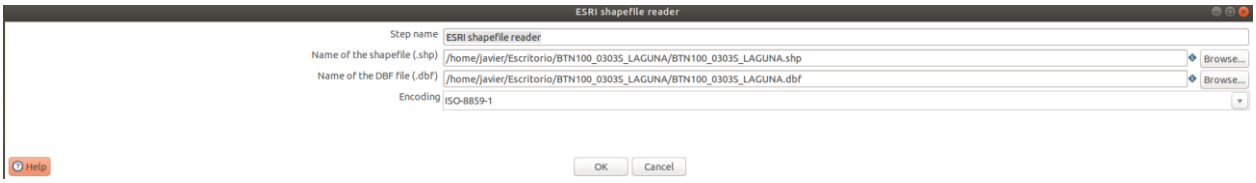

Figura 4: Cambio de rutas para los nuevos ficheros Shapefile.

Los inputs ya están arreglados. Ahora con el output: hay que crear un fichero .ttl que contendrá el resultado final de la transformación y sustituir el path antiguo por el nuevo. En este caso se ha creado el fichero laguna.ttl en el escritorio (véase la imagen inferior).

5 https://github.com/oeg-upm/btn100/tree/master/transformaciones-shape/3-Hidrografia/3-Laguna

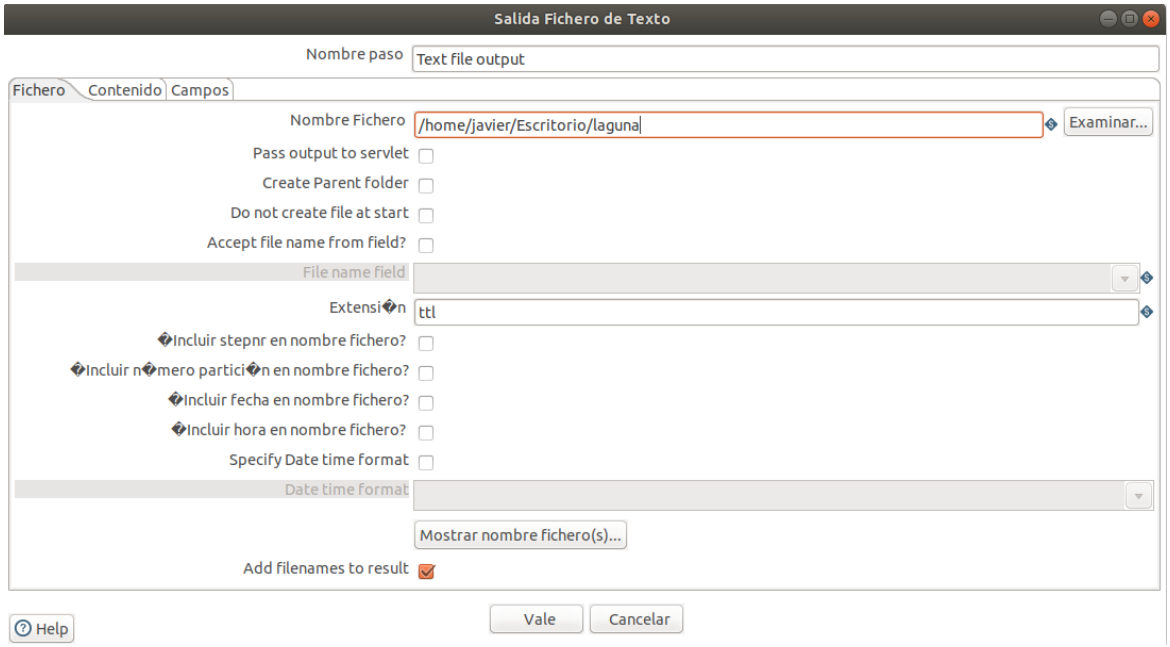

Figura 5: Cambio de rutas para el nuevo fichero Turtle.

Al ejecutar la transformación ahora, un error sigue saltando por consola (figura 6). Solucionar este error exige usar el debuger. Para ello habrá que crear un proyecto en Eclipse (se ha elegido este entorno porque viene por defecto en la configuración del ejemplo de la documentación de ayuda de Pentaho y porque personalmente vengo usando este entorno desde hace mucho tiempo y lo manejo bien) [8], y después seguir los pasos de la documentación para usar el debuger [9].

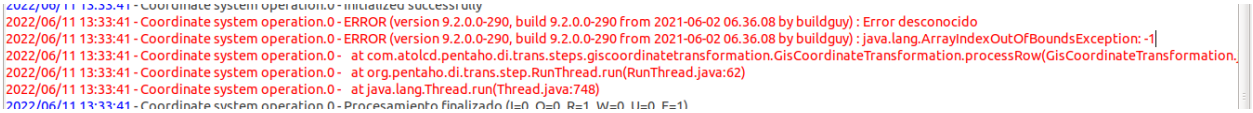

Figura 6: Error en la consola de Spoon.

Al intentar conectar el debuger salta algún tipo de error relacionado con la máquina virtual que no se ha sabido arreglar. Sin embargo, mientras se intentaba arreglar, se notó que el error viene de “Coordinate system operation”, una operación que forma parte del plugin de Pentaho. Esto levantó una sospecha: el error probablemente no venía de ahí (Pentaho es una rama de una empresa importante que ya tendrá muy bien estudiado su software). Así que se optó por rehacer el .ktr desde cero para ver dónde empezaba el error.

Construir el .ktr ha implicado revisar el fichero .shp y con ello descubrir el problema: todo va bien hasta “Value mapper” incluido. Al reescribir el paso “Coordinate system operation”, se vio que el campo “the_geom” no formaba parte del fichero .shp, lo que lógicamente daba lugar a una excepción. El fichero .shp dispone tan solo de los siguientes campos: filename; filetype; shapenr; partnr; nrparts; pointnr; nrpointS; x; y; measure; ID; ID_BD; ID_CODIGO; ID_MOD; FECHA_ALTA; REGIMEN; ETIQUETA. Por lo tanto, el fichero .shp contiene en cada fila un punto perteneciente a una laguna. Esto quiere decir que puede haber varias filas dedicadas a la misma laguna, ya que una laguna es un objeto geométrico “Polygon” que está formado por un conjunto de puntos.

Consecuentemente hay que generar un campo nuevo al que se llamará “the_geom” que contendrá por cada fila un objeto geométrico de tipo “Point”. En definitiva, una concatenación de los campos “x” e “y”, pero expresados en el formato adecuado (WKT/EWKT). El formato será entonces “POINT (x y)”. Para ello se ha intentado crear un .ktr que no ha dado resultado (figura 7) porque se ha usado en general la función “String operations” (figura 8) de Pentaho. Dicha función permite al usuario gestionar Strings, pero de una forma algo restrictiva: si el tamaño de la cadena varía, entonces no sirve porque hay que establecer el campo “Pad Length” que indica el número de caracteres a la derecha/izquierda (“Padding”) que hay antes de copiar la cadena “Pad char”. En un principio se pensó que el número de cifras de x e y era constante, pero al ejecutar la transformación es cuando se puede ver que realmente no es así.

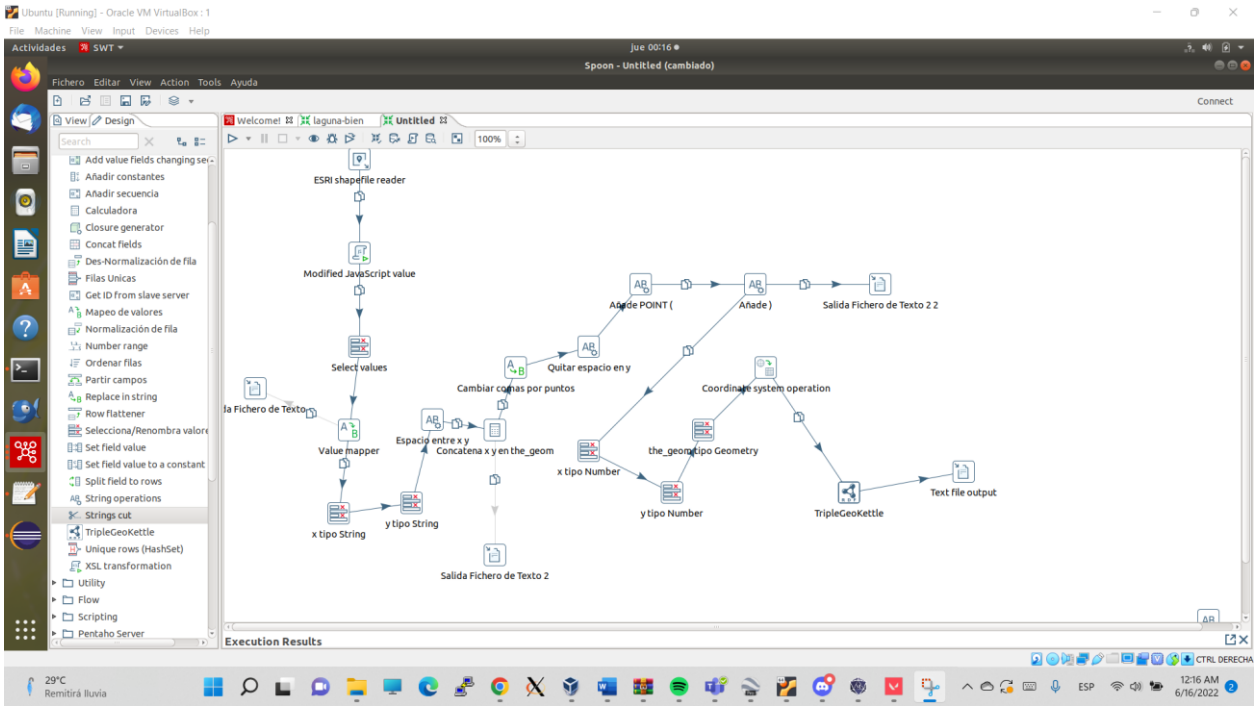

Figura 7: Primer intento de pipeline para generar "the_geom”.

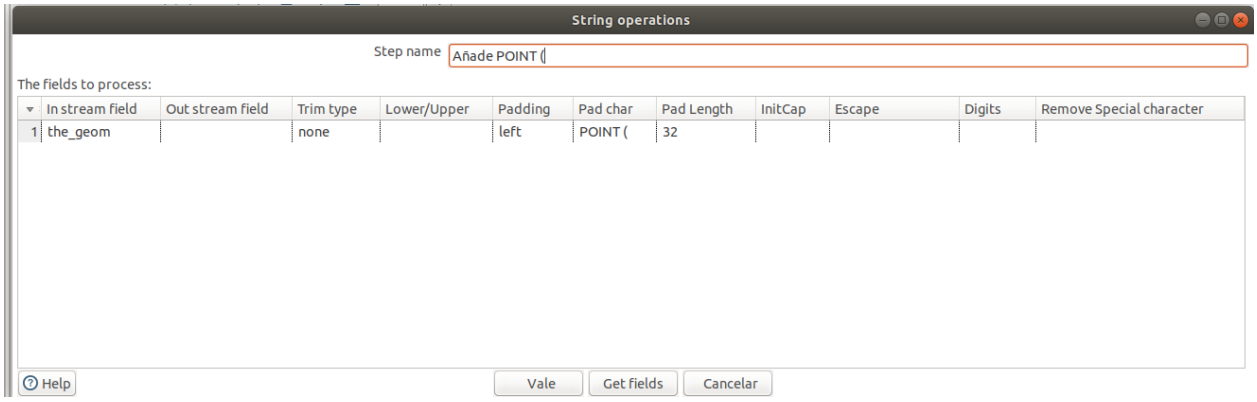

Figura 8: Contenido del paso “Añade POINT (”.

Es por ello por lo que hay que buscar otra alternativa para concatenar los Strings. De hecho, una forma mucho más sencilla y directa es utilizar un JavaScript (figura 9) que simplemente genera el String con los campos x e y. El

resultado del .ktr es el que se presenta en la figura 10. Ahora parece que el paso que antes daba problemas (Coordinate system operation) ya no los da. Sin embargo, en esta ocasión el error lo da el paso de TripleGeoKettle.

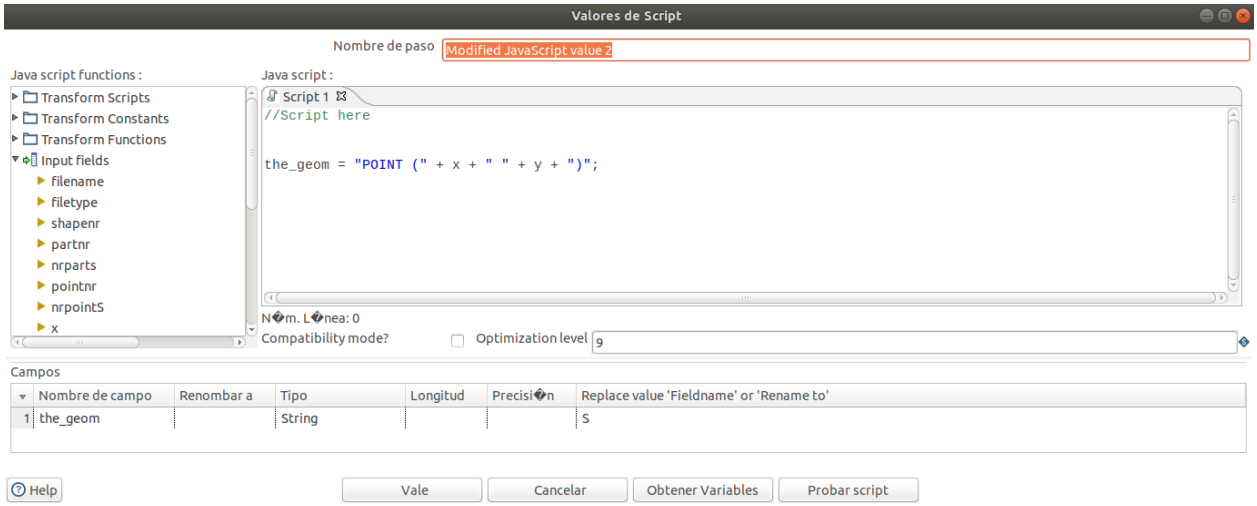

Figura 9: Generación de “the_geom” a través de un script.

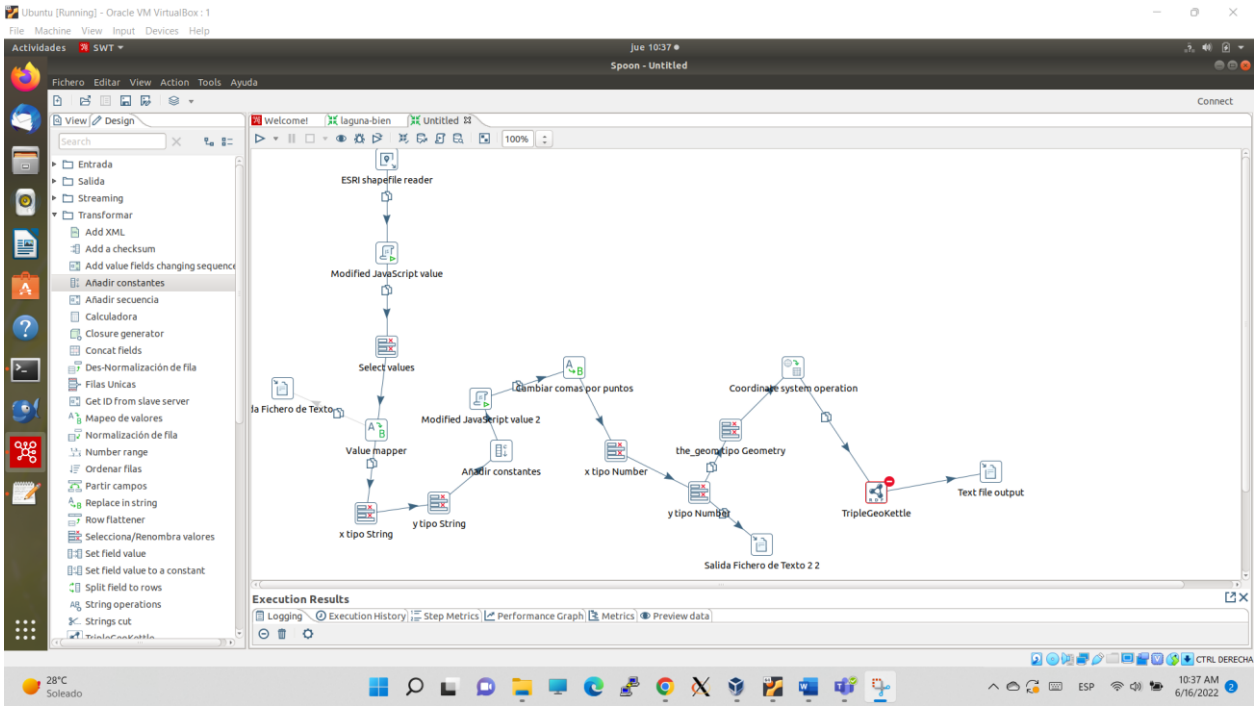

Figura 10: Segundo intento de pipeline para generar "the_geom” y error en TripleGeoKettle.

El error que devuelve TripleGeoKettle es una excepción: AttributeNotFound. Hay algo que no encaja; si se hace una pausa y detalladamente se comparan el paso “original” (izquierda figura 11) con el paso si se resetean las columnas (derecha figura 11) se puede apreciar que los campos no son parecidos. Por ejemplo, campos como “replaces”, “title” o “created”, que aparecen en la transformación original, no lo hacen en el Shapefile. Este discurso hace pensar

que el Shapefile que se utilizó (del repositorio del OEG) no coincide con el Shapefile que se usó en trabajos previos. Sin embargo, tras descargar el Shapefile directamente del portal abierto del IGN y usarlo en lugar del Shapefile del OEG, el resultado sigue siendo el mismo. Por lo tanto, el error no proviene del Shapefile. Hay que seguir adelante con el Shapefile original y modificar el .ktr para obtener el .ttl correspondiente.

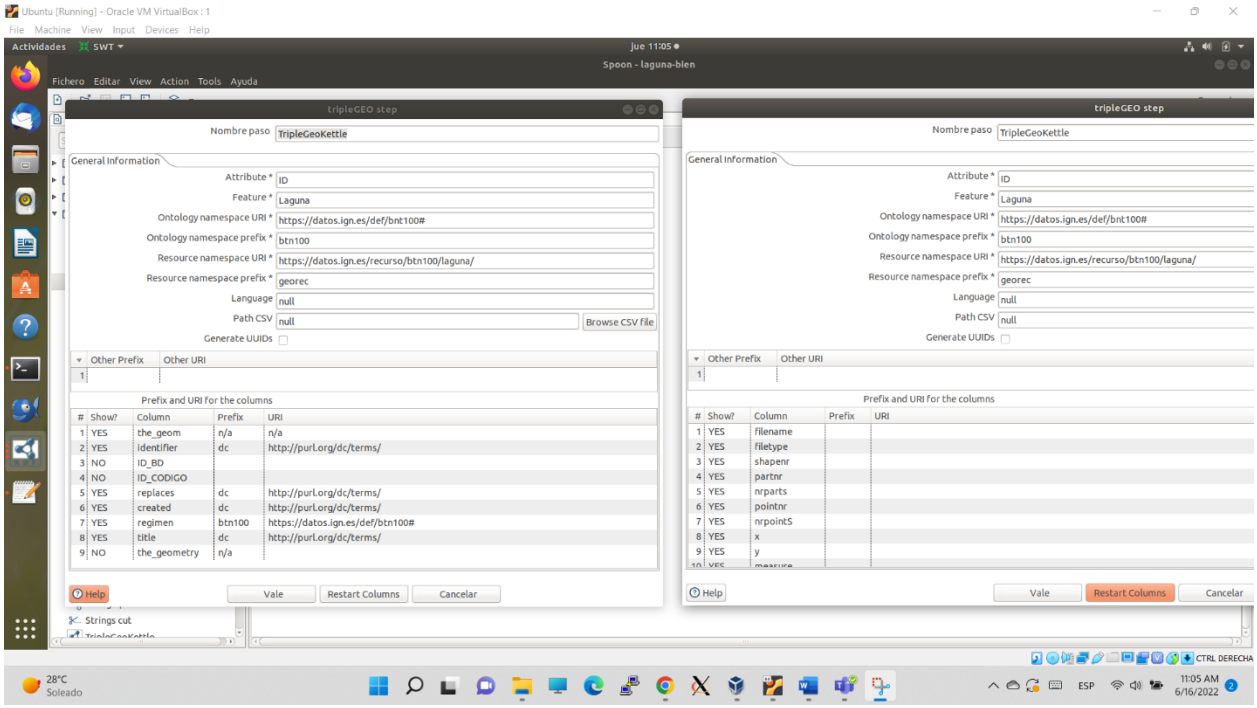

Figura 11: Comparación de los campos antes y después de resetear el flujo de entrada en TripleGeoKettle.

El resultado es el que se muestra en la figura 12, pero el fichero .ttl resultante sigue sin ser correcto del todo. El fichero .ttl que hay que obtener se muestra en la derecha de la figura 13, y sin embargo se obtiene el fichero .ttl de la izquierda de la figura 13. La estructura es correcta, pero falla la geometría. Si bien es cierto que un objeto geométrico de tipo POLYGON es un conjunto de puntos, su expresión en formato WKT es, tal y como se puede ver a la derecha, POLYGON ((x1 y1, x2 y2, x3 y3, … xn yn, x1 y1)), y el .ttl resultante no es correcto. La clave está en que en el Shapefile aparece por cada fila un punto perteneciente a una laguna, y se ha tratado a cada punto a parte, como si fueran objetos geométricos de tipo POINT independientes.

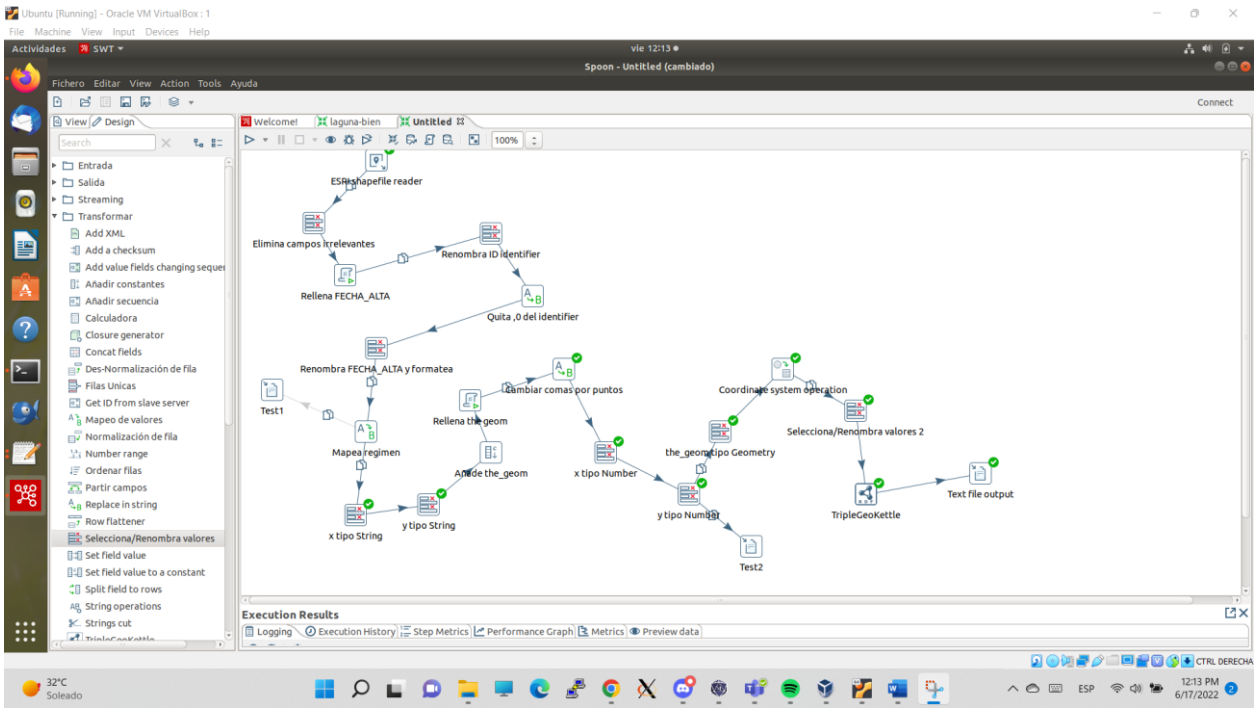

Figura 12: Pipeline que genera el Turtle con fallo en la geometría.

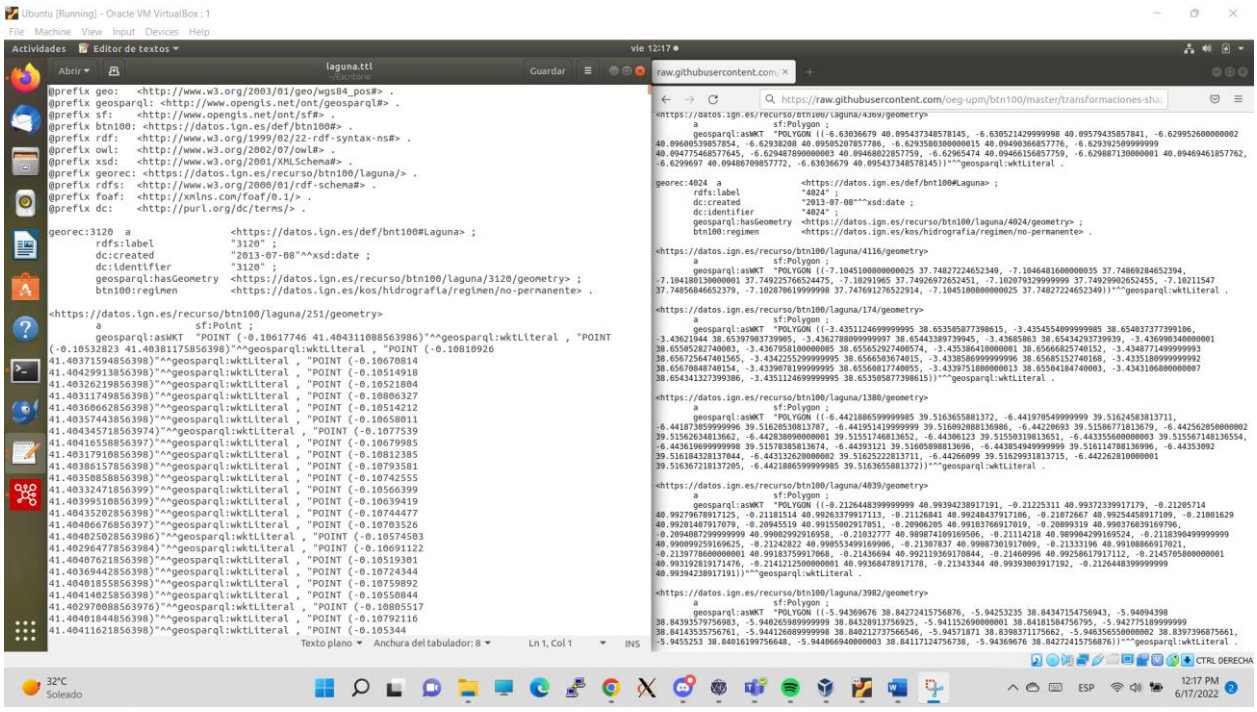

Figura 13: Comparación del Turtle correcto (derecha) con el generado erróneamente (izquierda).

La solución pasa por generar una entrada en la tabla por cada laguna, y que su geometría sea su conjunto de puntos formando un polígono. Para ello, en primer lugar, el contenido del campo the_geom se ha cambiado. Previamente las instancias tenían el formato “POINT (x y)”. Ahora el formato es “x y”. Después se hace uso de la función “Memory group by” para concatenar todos los campos

"the_geom" cuyos identificadores de lagunas sean iguales (separados por comas) y finalmente se le da el formato WKT con un script: POLYGON ((the_geom)). Un problema que surge es que hay ciertas erratas en el Shapefile: en teoría los puntos de cada laguna que aparecen en el Shapefile aparecen de manera ordenada; en primer lugar por el identificador de laguna y en segundo lugar por "nparts", que indica el orden de los puntos que forman la laguna. Sin embargo, al ejecutar la transformación, el paso en el que se cambia el tipo de "the_geom" de String a Geometry da fallos para algunas lagunas. Concretamente el fallo que resulta es que el polígono no es cerrado (recordar que un objeto geométrico de tipo POLYGON debe empezar y terminar con el mismo punto, de manera que la cadena de puntos sea cerrada). Indagando en el Shapefile y gracias a la traza de PDI se puede apreciar que para algunas lagunas (como por ejemplo la 1009), los puntos no aparecen en el orden correcto puesto que el primero no es igual que el último. Para eso está el tratamiento de errores en el paso del cambio de tipo de the_geom, que "ignora" el hecho de que no se respete el orden de aparición de los puntos. La solución se muestra en la figura 14 (véase el punto 2.4.2, en el que se aclara que realmente no hay erratas en los datos).

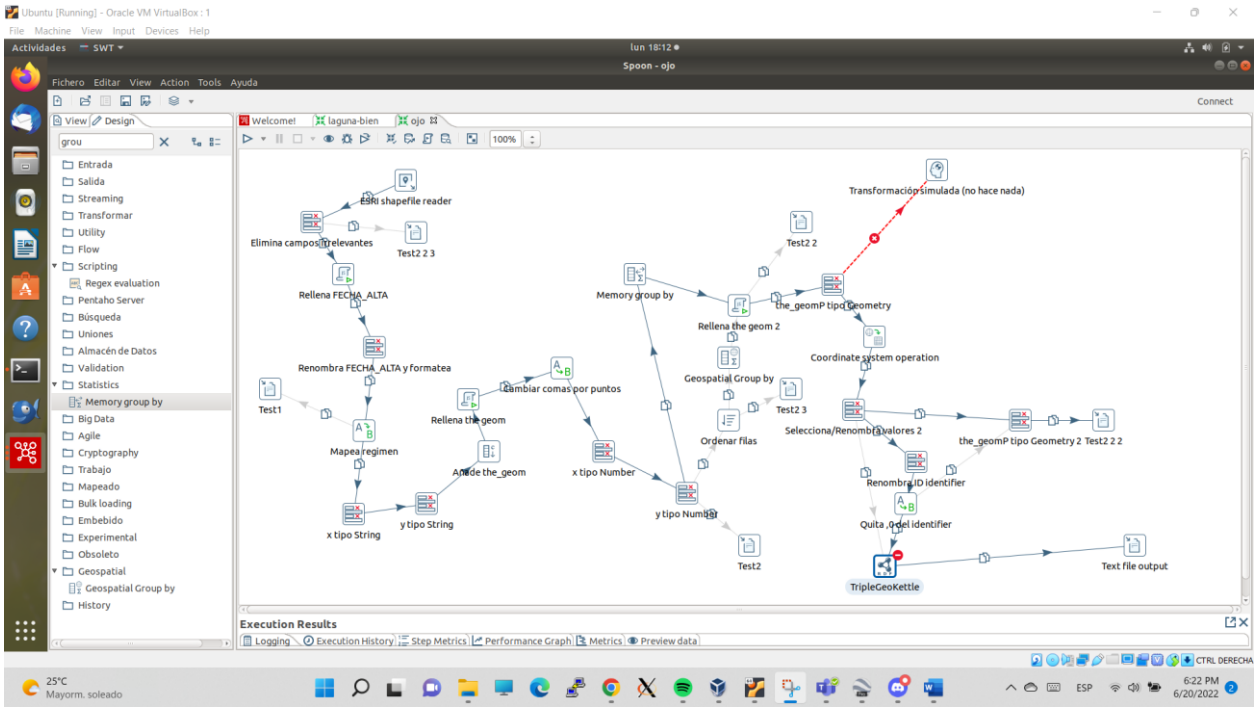

Figura 14: Pipeline final que genera correctamente "the_geom"

Ahora el error está en TripleGeoKettle: un null pointer (figura 15). Parece que el paso no es capaz de generar la tripleta con predicado rdfs:label. Sin embargo, antes de arreglar el problema con la geometría, no parecía haber problemas, y el paso TripleGeoKettle funcionaba. Si se presta atención, ocurre algo raro con este paso: antes de presionar el botón "vale", el resultado de resetear las columnas y rellenar los prefijos y las URIs adecuadamente es el que se muestra en la figura 16. Todo parece estar bien, pero al presionar "vale", si volvemos a fijarnos en el paso, la tabla cambia y pasa a ser la de la figura 17. En ella, el campo the_geom se duplica y desaparece el campo identifier (que es el objeto que debería formar parte de la tripleta cuyo predicado es rdfs:label).

Es posible que ese sea el motivo por el que salta el null pointer. ¿Pero por qué? ¿Por qué la tabla cambia al confirmar los cambios?

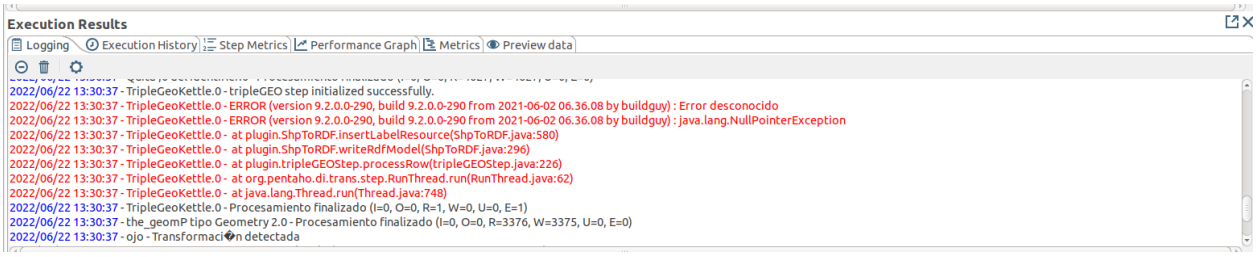

Figura 15: Error en la consola de Spoon.

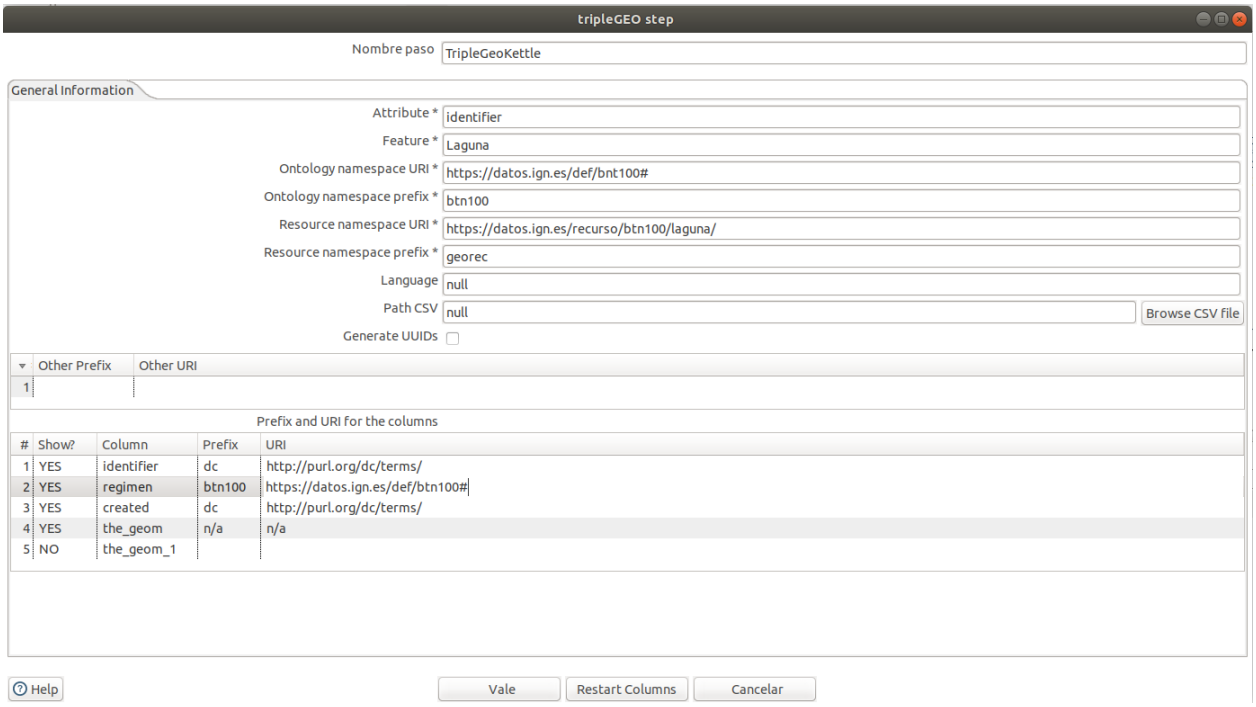

Figura 16: Edición del paso TripleGeoKettle.

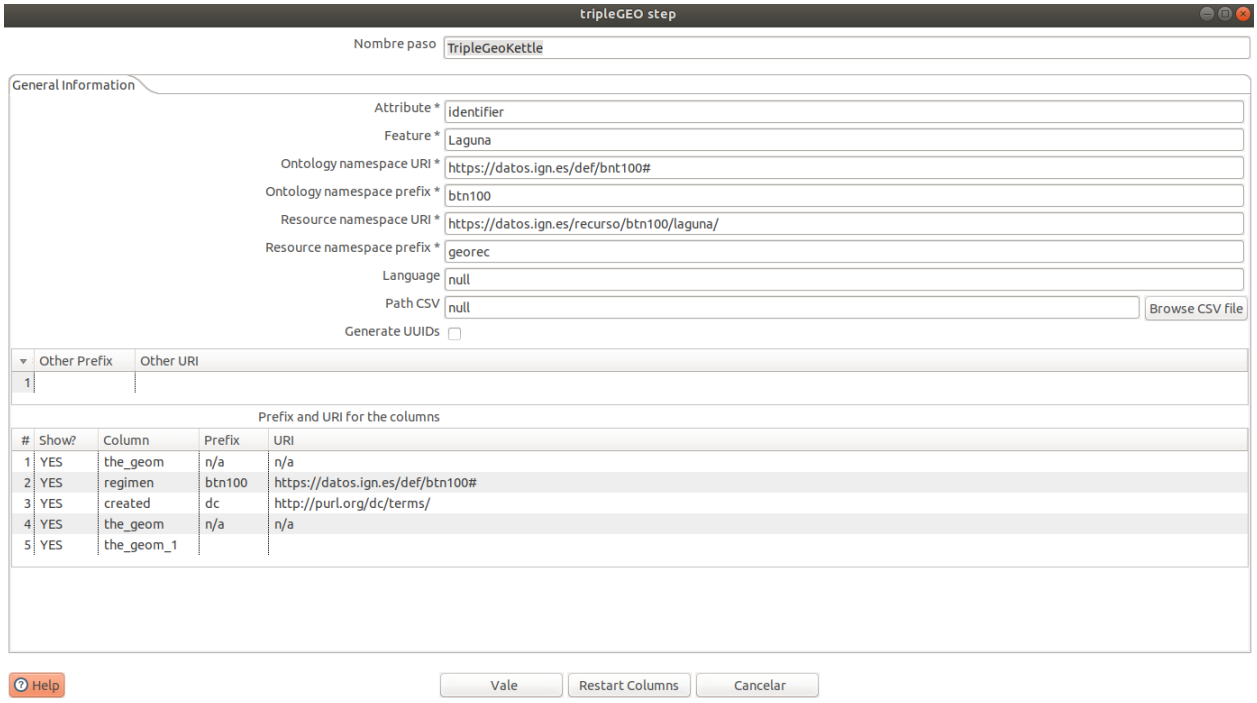

Figura 17: Edición del paso TripleGeoKettle.

Finalmente se ha notado que el orden en el que aparecen los campos en la tabla es relevante: parece que el campo “the_geom” tiene que ir a la fuerza en la primera posición, aunque al resetear las columnas el resultado no sea ese. Así que lo que se ha hecho es tomar las filas tal y como se muestra en la figura 16 e intercambiar de posición “identifier” con “the_geom”. Efectivamente, ahora al presionar “vale” y volver a revisar el paso, la tabla no cambia (figura 18). Al ejecutar la transformación ahora, sale otro error diferente: atributo no encontrado. Pero lo único que se ha hecho es cambiar las filas de orden… Si se vuelven a cambiar las filas de orden de tal forma que queden, en orden descendente, “the_geom”, “identifier”, “created”, “regimen”, “the_geom1”, entonces volvemos al mismo error de la figura 15.

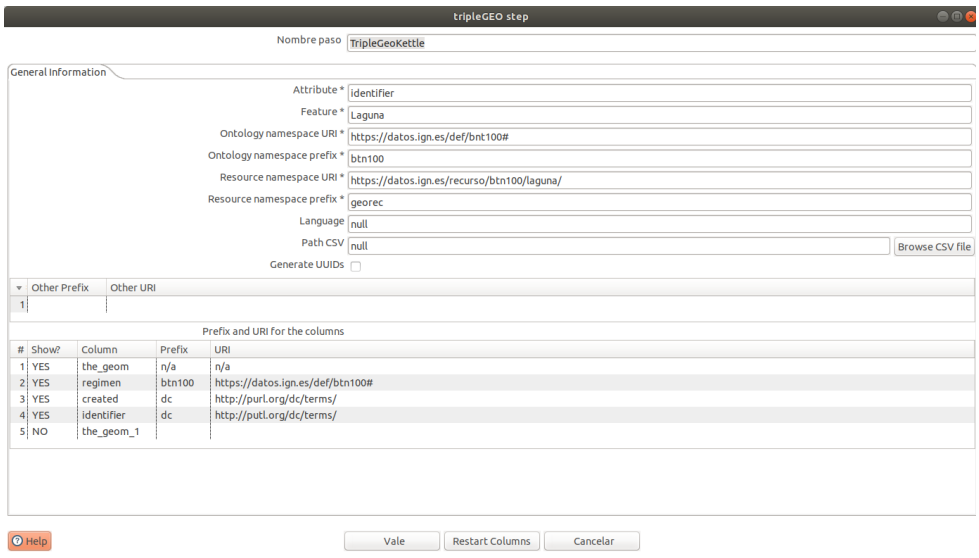

Figura 18: Edición del paso TripleGeoKettle

Después de intentar arreglar el error no se ha llegado a ninguna solución. Se ha llegado a un callejón sin salida. En conclusión, se ha generado un pipeline semi funcional partiendo de cero, que recoge un Shapefile y de forma manual y a través de sucesivas transformaciones, genera el campo “the_geom” en el formato adecuado (véase 2.4.2).

2.2 Capítulo 2: Generación del campo “the_geom” automáticamente y realización del pipeline de lagunas.

La motivación del desarrollo del pipeline en el capítulo 1 era que el campo “the_geom” (que en teoría debía generarse de manera automática) no se generaba. Después de investigar las causas, no se llegó a ninguna conclusión, así que se optó por generar el campo “the_geom” de forma manual. Al llegar al final del capítulo, se alcanzó un punto de inmovilidad, lo que ahora lleva a re investigar las causas por las que en un principio el campo “the_geom” no se generaba: esta vez sí que ha dado resultado la investigación.

El problema estaba en el lector “ESRI shapefile reader” (al cual se hace mención al final del apartado 1.3.3). Por motivos que se desconocen, este lector no genera el campo “the_geom”. No se ha encontrado ninguna referencia a este problema por parte de ningún otro usuario en internet. En trabajos pasados (como el de Beñat [10]) tampoco daba problemas. Finalmente la solución ha estado en cambiar el lector; en su lugar se ha utilizado “GIS File input” (figura 19). A diferencia del lector de ESRI (que recibía dos partes del Shapefile: .shp y .dbf), GIS solo recibe como entrada una parte del Shapefile: .shp. La configuración del paso es sencilla: añadir la ruta del .shp, escribir el nombre del campo que contendrá la geometría (en este caso “the_geom”) y concretar el encoding.

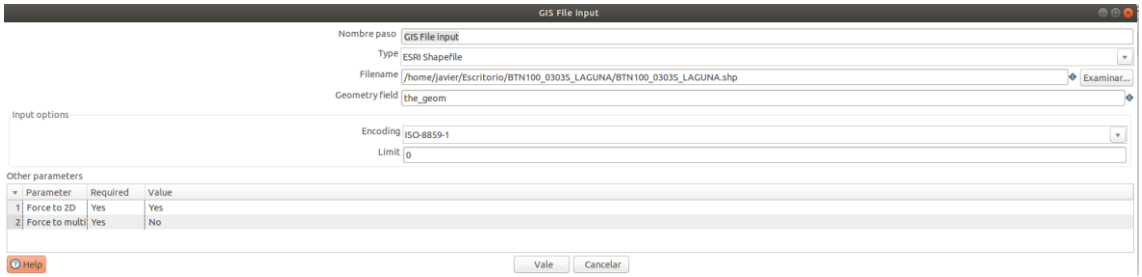

Figura 19: Paso GIS File input.

El resultado se muestra en la figura 20. Un .ktr que nace de laguna-tests.ktr[6], transformación a la que se han realizado ciertos cambios. A continuación procede una explicación del pipeline, detallando las funciones de cada paso: el primer paso ya ha sido explicado previamente (figura 19), recibe como input el .shp; el segundo paso, “Modified JavaScript value” formatea la fecha de creación de la laguna; “Select values” cambia el nombre de ciertos campos y devuelve la fecha de creación al tipo original “Date”, ya que en el paso anterior, con el fin de formatear las fechas, se cambió el tipo a “String”; “Value mapper” cambia en el campo “regimen” los unos por “https://datos.ign.es/kos/hidrografia/regimen/permanente” y los doses por “https://datos.ign.es/kos/hidrografia/regimen/no-permanente”; “Coordinate system operation” reproyecta la geometría, que en un principio está en

6 https://github.com/benatagirrea/soporte-GeoPackage-herramientas-OEG/tree/main/transformaciones

ETRS89, y la pasa a WGS84; y finalmente “TripleGeoKettle” recoge el flujo de entrada y genera el .ttl con las URIs correspondientes.

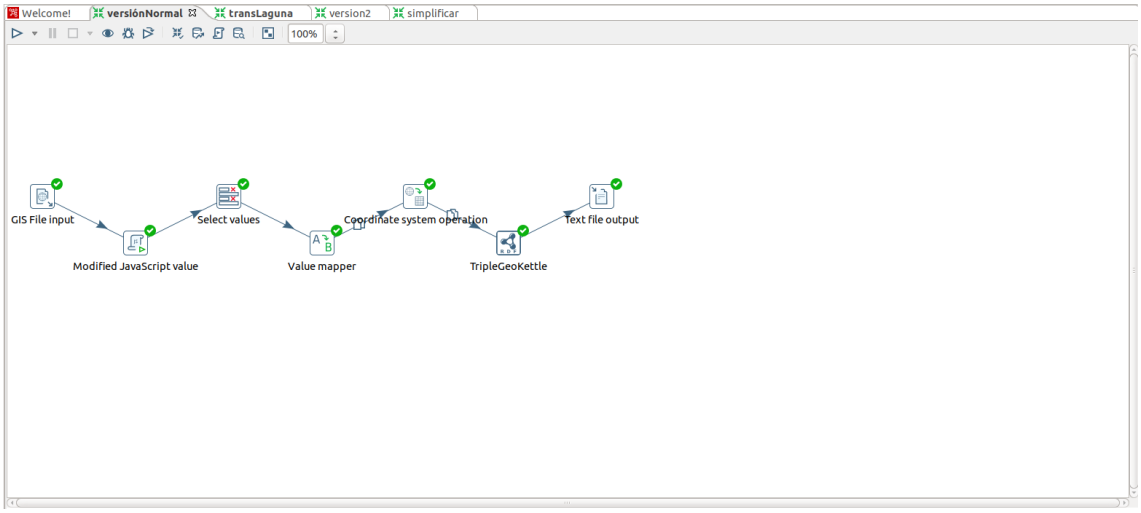

Figura 20: Pipeline que genera el recurso laguna.ttl.

2.3 Dificultades encontradas.

2.3.1 Determinismo PDI.

Un pequeño detalle que se ha pasado por alto durante el inicio del proyecto es que PDI mueve todos los pasos de forma concurrente. Cuando un paso recibe filas, trabaja con ellas, sin esperar a que lleguen todas. Esto se traduce en que la ejecución de la misma transformación puede llevar a resultados ligeramente diferentes en la traza, porque la velocidad de los pasos no es constante (la máquina puede estar ocupada con otras tareas, lo que implica más lentitud en la transformación). Este detalle ha llevado a retrasos porque para mí, transformaciones que no deberían alterar el resultado entre ejecuciones diferentes, sí que lo hacían: en ocasiones se llegaron a utilizar pasos bloqueantes con el fin de controlar el flujo ("Block this step until steps finish" o "Blocking step") intentando "corregir" lo que realmente funcionaba pero que se pensaba que fallaba. Finalmente investigando se encontró esta característica de PDI tan peculiar.

2.4.1 Confusión por pasos inútiles.

Otro problema que ha surgido debido a mi inexperiencia es el conjunto de pasos de la transformación base laguna-tests.ktr cuya funcionalidad era nula. Causaron cierta confusión porque los pasos están definidos en la transformación, pero no se utilizan. Para alguien que nunca ha manejado PDI y que no entiende bien el porqué de ciertas cosas, estos pasos no hacen más que distraer y desviar la atención hacia temas de importancia nula. En concreto se trata de los pasos "shp input", "gpkg input", "gpkg input 2" y "TripleGeoKettle.gpkg" (refiriendo a la figura 2).

2.4.2 ¿Erratas en datos del IGN?

En primera instancia se pensó que había ciertas erratas en los datos de algunas lagunas. Sin embargo, más adelante se notó que en realidad no había erratas. Para explicarlo, se expone en el primer párrafo la postura inicial (e incorrecta), para pasar a explicar la fuente del error que se tuvo en el segundo párrafo.

En el capítulo 2 del desarrollo no hubo problemas con esto, probablemente porque el lector de SIG no tenga en cuenta o no se cerciore de que la geometría sigue el formato WKT, pero durante el desarrollo del primer capítulo sí que se hizo notar que había erratas en la representación poligonal de algunas lagunas. Al hacerlo de forma manual, cuando se cambia el tipo de un campo (en este caso "the_geom") a tipo "Geometry" a través del paso "Select values" sí que se comprueba internamente que el formato sea correcto, lo que ha dado problemas. En un principio se barajó la posibilidad de corregir estas erratas, pero después de pensarlo detenidamente se llegó a la conclusión de que eso no era posible por un motivo; y es que se puede hacer que el primer punto y el último sean iguales (de hecho, entre los puntos que forman el polígono se puede ver que el

primer punto de la serie vuelve a salir entre medias), pero nada asegura que el orden de los puntos sea correcto. Una laguna puede tener una geometría irregular y no hay forma de conocer el orden correcto de los puntos.

El razonamiento de que el formato WKT para un polígono ha de empezar y terminar por el mismo punto siempre no es correcto del todo. Un objeto de tipo POLYGON no tiene por qué ser una lista de puntos que forman una cadena cerrada; sino que puede ser una lista de listas de cadenas cerradas. Es decir, que la regla de que la cadena ha de ser cerrada (que el primer punto y el último sean iguales) no se aplica al polígono en general, sino a cada cadena de puntos que forman el polígono. El sentido que tiene que un polígono esté formado por varias cadenas es que el polígono puede tener huecos. Imaginemos por ejemplo una laguna que rodea una montaña. Si se quiere excluir la montaña de la laguna, habrá que definir en primer lugar una cadena cerrada de puntos que represente la laguna, y después otra cadena de puntos que represente la montaña. De esta forma se obtiene un polígono que tiene un hueco en medio (como una rosquilla, si se me puede permitir).

Durante el primer capítulo del desarrollo, al generar el campo “the_geom” manualmente no se tenía en cuenta este detalle. Los ficheros de texto salientes que se utilizaban para testear simplemente mostraban los puntos de la laguna, pero no había indicaciones extra acerca de si el polígono estaba formado por varias cadenas. La manera en la que se generaba el campo era a través de “Memory group by”; se concatenaban los puntos que pertenecieran a la misma laguna, y separados por comas, desde el primero hasta el último, sin tener en cuenta lo comentado anteriormente, lo que daba lugar al error que se comenta durante el desarrollo del capítulo 1.

Véase la figura 21, que muestra la geometría de la laguna 1009. Como se puede apreciar, el primer punto de la serie se repite entre medias, pero justo después hay un paréntesis, que marca el final de la primera cadena. Después hay otra cadena que también empieza y acaba por el mismo punto. En este caso, el polígono es del tipo “POLYGON ((x1 y1, x2 y2, … xn yn, x1, y1), (xa ya, xb yb, … xa ya))”.


```
<https://datos.ign.es/recurso/btn100/laguna/1009/geometry>
        a               sf:Polygon ;
        geosparql:asWKT  "POLYGON ((3.1167290599999995 39.8307196885434, 3.11692273 39.83055798854339, 3.1177750500000005 39.83029866854341, 3.11808725 39.8300628785434, 3.1181711900000013
39.829917198543406, 3.11829814 39.829738688543415, 3.1184139900000005 39.82942316854342, 3.1185591000000006 39.82932213854341, 3.1194933100000006 39.82914802854342, 3.1194542000000003 39.829087898543406,
3.1194073000000001 39.829006898543426, 3.1193794600000001 39.828968348543405, 3.11877546 39.82912223854343, 3.1184972300000001 39.82911174854342, 3.11817982000000 39.82897333854341, 3.1178977099999994
39.828752808543406, 3.11765392 39.828622638543415, 3.11715723 39.82856302854342, 3.11640796 39.82871673854342, 3.11606546 39.82874022854342, 3.1152388800000006 39.82894868854343, 3.1151393999999999
39.82898971854342, 3.11485222 39.828677128543426, 3.1145728200000002 39.82786146854343, 3.114155689999999 39.82726834854341, 3.114107989999999 39.82641201854344, 3.1141298600000002 39.825644818543445,
3.1139665400000003 39.82482427854346, 3.11373312 39.82359345854346, 3.113521789999999 39.82172032854349, 3.1131263200000006 39.820418298543494, 3.1125455299999993 39.8189737185435, 3.11191925
39.817993048543514, 3.11173375 39.81774344854354, 3.1110161 39.81747651854353, 3.1106243600000005 39.818458168543515, 3.110973040000001 39.8194747885435, 3.11079064 39.82115204854349, 3.11060717
39.82216917854349, 3.1103753899999997 39.82191961854348, 3.11060752 39.82030995854354, 3.1106603100000005 39.81994387854349, 3.1102327999999995 39.819564698543495, 3.109909280000001 39.81981478854348,
3.10951805 39.82113541854349, 3.10944988 39.821920498543484, 3.1097519799999995 39.822740908543466, 3.1098463000000005 39.823900508543474, 3.1097779 39.82448933854345, 3.1105669000000002
39.82591588854345, 3.1103828099999995 39.826558348543436, 3.1105691899999997 39.82734318854342, 3.1110557500000007 39.82773523854344, 3.110362410000001 39.828271138543435, 3.11008588 39.828985048543416,
3.1099244900000005 39.82935986854341, 3.1097257599999995 39.829405908543414, 3.10960076 39.82950290854341, 3.1094854799999996 39.82977062854341, 3.1089183500000006 39.82992427854341, 3.1085602599999995
39.8300212785434, 3.10814371 39.83000382854341, 3.107939040000001 39.829983538543416, 3.1076809499999993 39.83003993854341, 3.10760254 39.8303218785434, 3.1077221099999996 39.830489638543405,
3.1080288400000007 39.83053917854341, 3.10820914 39.8305087085434, 3.10832939 39.8303604785434, 3.10897201 39.83014445854341, 3.10960144 39.82993109854341, 3.109948629999999 39.8299842885434,
3.1101356499999997 39.82995023854342, 3.11022608 39.82984129854341, 3.11024098 39.829608138543406, 3.11010382 39.82942466854341, 3.1108713800000007 39.82819928854342, 3.111194869999999 39.82791351854344,
3.1116348200000004 39.82809150854343, 3.1117522500000003 39.82916186854341, 3.1129800799999994 39.830017058543405, 3.1121952499999996 39.83121318854339, 3.1121854 39.831401158543339, 3.112150359999999
39.83206960854338, 3.1121062800000001 39.83342558854336, 3.11175982 39.83383627854337, 3.11189922 39.834175128543365, 3.1123623200000003 39.834335248543375, 3.11240988 39.83512021854335, 3.10998136
39.83599676854335, 3.1099556399999995 39.83593501854335, 3.108889520000001 39.83337512854338, 3.1078710699999994 39.83323335854337, 3.107249189999999 39.83512510854335, 3.10910265 39.83644362854336,
3.110213099999999 39.836139278543335, 3.11234075 39.83529869854337, 3.11258301 39.83545911854336, 3.11271123 39.835423228543355, 3.1125750400000003 39.83398176854336, 3.1148841 39.833922428543374,
3.11482386 39.83366638854337, 3.112801 39.83371037854338, 3.1124764599999994 39.833371708543375, 3.112385819999999 39.8314099685434, 3.11298063 39.8303560485434, 3.1133042799999995 39.83015947854339,
3.114461920000001 39.830533008543405, 3.11564218 39.8305853585434, 3.1162794700000007 39.830949608543406, 3.11655795 39.830965428543394, 3.1167290599999995 39.8307196885434), (3.11524389
39.82920350854341, 3.11548373 39.829108948543414, 3.1161809299999996 39.82902466854341, 3.1165094000000004 39.82891490854343, 3.1170136600000005 39.82876442854342, 3.1174591899999995 39.82876934854342,
3.11789669 39.828998938543414, 3.1181493399999995 39.82935366854341, 3.1181012199999993 39.829707948543415, 3.1178720799999993 39.830046318543396, 3.1176634600000006 39.83017533854339, 3.11684849
39.8303397385434, 3.1166248400000005 39.8303925285434, 3.11594289 39.8305136985434, 3.1156649600000006 39.83037123854339, 3.11524389 39.82920350854341))"^^geosparql:wktLiteral .
```


Figura 21

Debido a este problema, el desarrollo del capítulo 1 jamás hubiese llegado a nada. Como mucho se hubiera obtenido un fichero .ttl con algunas geometrías mal definidas.

3 Resultados y conclusiones

Finalmente, aprender a usar PDI y gestionar las transformaciones ha sido más complicado de lo esperado. Es por ello que el objetivo principal del proyecto ha resultado ser demasiado ambicioso y en consecuencia no ha sido posible alcanzarlo. Sin embargo, pese a no haber logrado enlazar los datos a través de su geometría, se han obtenido nuevos conocimientos en el ámbito. Unos conocimientos fundamentales que en un futuro se podrían explotar para llegar aún más lejos.

Fundamentalmente se han alcanzado los siguientes objetivos:

- Aprendizaje de herramientas y métodos actualizados en el contexto de la información geográfica.
- Desarrollo de un "pipeline" de datos en PDI semi funcional desde cero.
- Desarrollo de un "pipeline" de datos en PDI completamente funcional a partir de trabajos previos.

En lo personal, me gustaría haber llegado más allá. Tenía la ambición de conseguir enlazar datos geométricos, pero mi única base para la realización de este trabajo era haber cursado "Semantic Web, Linked Data and Knowledge Graphs". Al principio pensé que sería un buen punto de partida, pero al empezar a tratar con conceptos tan nuevos para mí como PDI o SIGs me di cuenta de que no era suficiente. Prácticamente ha sido aprender desde cero a utilizar herramientas; nunca es fácil arrancar con cosas nuevas. Y para mí tampoco.

Para concluir, mis esfuerzos durante el TFG se han enfocado a transmitir un proceso de aprendizaje, y he reflejado mi viaje a través de dicho proceso. He trabajado para conseguir que las bases queden bien definidas; así, más estudiantes podrán recurrir a este documento al empezar a tratar con PDI y SIGs en un futuro. Porque factores que ahora veo claros, en un principio nunca lo fueron; como por ejemplo darse cuenta de que el campo "the_geom" se tenía que generar de forma automática, y que el lector de ESRI no lo hacía. He invertido demasiado tiempo en el desarrollo del capítulo 1, un tiempo que podía haber aprovechado para cumplir los objetivos iniciales si hubiera sabido desde el principio lo que sé ahora.

3.1 Línea futura

Aunque el capítulo 1 del desarrollo no haya resultado, sería posible continuar investigando y tratando de solucionar el problema para crear un pipeline que emulase la elaboración de la geometría a través de transformaciones de PDI, que es realmente lo que se intentó. Para continuar habría que, en primer lugar, rectificar la creación de los polígonos (en el punto 2.4.2 se explica en detalle este problema), y en segundo lugar arreglar el error en el plugin de TripleGeoKettle. De hecho, incluso se podría extrapolar no solo a polígonos, sino también a otras estructuras, tal y como lo hace el lector de GIS. De esta manera se obtendría un emulador completo para generar el campo de geometría.

4 Análisis de Impacto

4.1 Personal

La realización del trabajo ha supuesto para mí no solo aprender acerca de otro tema nuevo, sino también la gestión de problemas inesperados y sobre todo a adaptarme a la situación. Durante la carrera siempre se mandan trabajos, y pese a que pueda haber diversas formas de resolverlos, la meta es la misma. No recuerdo haber empezado un trabajo con una idea tan distinta a la que finalmente se ha alcanzado. Pero para nada es algo malo; se han alcanzado otros objetivos, que servirán a otras personas.

Además, para mí este trabajo es el último de la carrera. Personalmente estoy satisfecho con mi trabajo a lo largo de estos 4 años y sobre todo feliz por haber llegado hasta aquí.

4.2 Empresarial

PDI debería tomar nota acerca del bug encontrado con ESRI shapefile reader y solucionarlo. El trabajo no incluye la solución, pero sí que consta el problema como tal.

4.3 Social

Me atrevería a decir que el impacto social será el más grande. El enfoque final del trabajo ha sido didáctico. La intención principal es la de ayudar a futuros estudiantes o nuevos investigadores a que sus inicios en estas tecnologías sean más sencillos. Esto se ha conseguido estableciendo unas bases firmes; gestión de plugins, uso básico de Spoon PDI…

El impacto social ha sido la base en la toma de la decisión más importante en el trabajo: pasar de enlazar datos a través de su geometría a enseñar el manejo de herramientas de forma asequible para cualquier persona interesada. En un principio se pretendía generar un impacto más empresarial/económico, pero finalmente se ha optado por generar un impacto social. Haciendo referencia a los Objetivos de Desarrollo Sostenible (ODS), se podría decir que el trabajo favorece los puntos 4 (educación de calidad) y 8 (trabajo decente y crecimiento económico), siendo especialmente importante el 4.

5 Bibliografía

[1] ¿Qué son los datos ráster?

"https://desktop.arcgis.com/es/arcmap/latest/manage-data/raster-and-images/what-is-raster-data.htm"

[2] Introducción a los SIG: sistemas vectoriales

"https://www.geogra.uah.es/gisweb/1modulosespanyol/IntroduccionSIG/GISModule/GIST_Vector.htm#:~:text=Un%20SIG%20vectorial%20se%20define,se%20asocian%20sus%20valores%20tem%C3%A1ticos."

[3] Página oficial de Hitachi

"https://www.hitachivantara.com/en-us/home.html"

[4] Página oficial del IGN

"https://www.ign.es/web/ign/portal"

[5] Información extra acerca del formato WKT

"https://es.wikipedia.org/wiki/Well_Known_Text"

[6] Página de descarga de PDI

"https://sourceforge.net/projects/pentaho/"

[7] Página de descarga del plugin "pentaho-gis-plugins"

"https://github.com/atolcd/pentaho-gis-plugins/releases"

[8] Documentación de ayuda para crear proyecto en Eclipse

"https://help.hitachivantara.com/Documentation/Pentaho/8.0/Developer_Center/PDI"

[9] Documentación de ayuda para usar el debuger

"https://help.hitachivantara.com/Documentation/Pentaho/5.4/0R0/0V0/010/040"

[10] TFG de Beñat

"https://github.com/benatagirrea/soporte-GeoPackage-herramientas-OEG"