Universidad Autónoma de Baja California
Instituto de Ingeniería

Tesis:
"Desarrollo de una arquitectura de software para la descarga, agregación geoespacial y consulta de datos provenientes de estaciones climáticas"

Presenta:
L.S.C. Luis Alejandro Herrera León

Para obtener el grado de:
Maestría en Ingeniería

Director de tesis:
Dr. Gabriel López Morteo

Codirector de tesis:
Dr. Francisco Munoz-Arriola

Mexicali, Baja California 12 de Junio del 2018

Capítulo 1. Introducción

Actualmente la cantidad de información generada es masiva, IBM (2012) reportó que se generaban 2.5 mil millones de gigabytes al día con una tendencia al crecimiento; de acuerdo con Zikopoulos et al. (2012), la cantidad de información almacenada será alrededor de 35 zettabytes para el 2020. Para poner esto en perspectiva un zettabyte es igual a 1 billón de gigabytes.No obstante que los datos disponibles crecen, la generación de información útil y utilizable se pueden ver menguados por la falta de herramientas de síntesis y análisis. En este documento busca crear una herramienta que permita al usuario de baja experiencia en manejo de datos heterogéneos y masivos (colección, transformación y almacenamiento) transformar datos climatológicos en información.

Existen tendencias hacia los datos abiertos por parte de organizaciones de México y Estados Unidos, *Open Knowledge Foundation* los define como: “datos que pueden ser utilizados, reutilizados y redistribuidos libremente por cualquier persona, y que se encuentran sujetos, cuando más, al requerimiento de atribución y de compartirse de la misma manera en que aparecen”. Los gobiernos en México y Estados Unidos han promovido nuevas políticas para brindar los datos abiertos a la sociedad y de esta manera fomentar la transparencia. La motivación de este estudio proviene de la alta cantidad de datos abiertos que no han sido explotados, debido a que muchos de los expertos en países en desarrollo como México no cuentan con los recursos o conocimientos computacionales para procesar y almacenar los datos.

Usualmente las plataformas de datos abiertos cómo http://data.gov en Estados Unidos o http://datos.gob.mx en México ponen a disponibilidad los datos en formatos y/o estructuras heterogéneas. Los datos pueden presentar las siguientes diferencias entre los conjuntos de datos definidas por Leser (2000):

- Técnica

  La fuentes de datos pueden diferir en la plataforma de hardware utilizada, el sistema operativo, lenguaje de consultas, mecanismo de acceso, representación de datos, etc.

- Modelo de datos

  Las fuentes de datos pueden presentar los datos utilizados un modelo orientado a objetos, semántico, jerárquico o relacional.

- Semántica

  Los datos pueden diferir en el significado de términos y unidades, lo que lleva a sinónimos y homónimos.

- Estructural

  Los datos pueden ser almacenados en diferentes estructuras. Por ejemplo diferentes grados de normalización.

- Esquemática

  Un caso especial de la heterogeneidad estructural. Si los datos son representados por diferentes conceptos del mismo modelo de datos, por ejemplo, atributo contra relación o valor contra relación.

Los científicos deben ser capaces de enfrentar la heterogeneidad que presentan los datos, por lo tanto si se presentan diferencias técnicas en los datos, el científico deberá contar con el conocimiento necesario para procesar los distintos formatos de

los archivos de datos. Si estas diferencias se encuentran en el modelo de datos se deben comprender las estructuras de los mismos así como su relación, y por último, si se presentaran diferencias semánticas se debe comprender las interpretaciones independientes por parte de los desarrolladores. Esta etapa de manejo de datos por parte del científico puede extender considerablemente la investigación que se desea realizar, dependiendo de los conocimientos y recursos computacionales con los que se cuenta.

Este problema se puede identificar en el campo de la meteorología donde los conjuntos de datos que se generan a lo largo del mundo presentan características heterogéneas, ya que estas pueden provenir de estaciones meteorológicas, satélites o derivados de procedimientos; estos orígenes generan productos diferentes, por ejemplo las estaciones meteorológicas usualmente generan productos puntuales mientras los satélites generan productos en malla. Del mismo modo en productos con el mismo origen se pueden presentar productos distintos, por ejemplo las estaciones *EddyPro* generan un formato de archivo llamado GHG mientras las estaciones *AcuRite* generan el formato de archivo CSV. Además otros formatos de archivos utilizados por este campo como: CSV, KML, TXT, NetCDF, JSON, GRD, GHG, entre otros. Por estas características se decidió analizar y procesar conjuntos de datos en este campo, algunos de los conjuntos utilizados fueron: Red Climática Histórica Global (GHCN por siglas en inglés) y conjunto de datos generado por una estación *EddyPro* en Queretaro.

Se utilizaron casos de estudio con una validación técnica para verificar que efectivamente al implementar directrices establecidas se puede integrar un nuevo conjunto de datos. Las directrices fueron establecidas a través de la implementación de una arquitectura de software basada en *plugins* para desarrollar una plataforma de integración de conjuntos de datos.

Justificación

El estudio realizado por la Corporación Internacional de Datos (IDC por sus siglas en inglés) en el 2012 menciona que sólo el 3% de todos los datos se encuentran actualmente etiquetados y listos para su manipulación, y sólo una sexta parte o 0.5% es utilizado para su análisis. Con estas cifras nos podemos percatar del conocimiento potencial sin explotar que se encuentran en esos datos sin analizar. A causa de todo este potencial surge el término “ciencia de datos”, que es descrita por Provost y Fawcett (2013) como un conjunto de principios fundamentales que apoyan y guían la extracción de información y conocimiento de los datos. Al ser esta una ciencia especializada, existe una demanda por científicos de datos, quienes deben poseer conocimientos en programación/software, estadística, matemáticas y comunicación para ser aplicados en áreas de conocimiento generadoras de datos.

Mientras existen altas demandas de científicos de datos para analizar los datos generados por las organizaciones, en la academia los investigadores que desean analizar datos para los estudios que realizan deben contar con algunos de los conocimientos en ciencia de datos, esto para ser capaces de obtener la información a partir de los datos. En el caso de los investigadores que no cuentan con

conocimiento de programación/software, se hace uso de herramientas de hojas de cálculo como Microsoft Excel, o bien, herramientas de análisis estadístico como IBM SPSS. Este tipo de herramientas cuentan con la capacidad de abrir un número finito de formatos de archivos, a causa de esto los investigadores que las utilizan deben asegurar que los datos que estén utilizando se encuentren en un formato de archivo y estructura que estas herramientas pueden utilizar y aquí es donde surge el problema de heterogeneidad de los datos.

Podemos citar el caso del Programa para la Evaluación Internacional de Alumnos (PISA por sus siglas en inglés), de la Organización para la Cooperación y el Desarrollo Económicos (OECD por sus siglas en inglés), este programa es una examen internacional trianual con el propósito de evaluar los sistemas educativos alrededor del mundo, al poner a prueba el conocimiento y habilidades de los estudiantes de quince años de edad. La OECD pone a disposición de todo el público los datos obtenidos en las evaluaciones realizadas. En su estudio realizado en el año 2012 estos datos se separaron en cinco categorías: estudiantes, escuelas, padres, objeto cognitivo y objeto cognitivo calificado; cada categoría cuenta con un documento ASCII con los datos, un archivo de códigos PDF, un archivo SPSS y un archivo SAS. El problema de heterogeneidad se presenta al desear comparar los distintos años, ya que no existe un estándar en la estructura de los datos PISA. Es decir, si un pedagogo que no cuenta con licencias para las herramientas IBM, SPSS o SAS desea analizar los datos de un año en particular deberá utilizar el archivo de códigos PDF, que detalla el significado de los datos para separar las columnas del

archivo ASCII, este procedimiento deberán seguirlo por cada año que desee consultar ya que el formato no es el mismo.

La resolución de este problema puede brindar a los investigadores la oportunidad de facilitar el manejo de los datos, de tal manera que el desarrollo de su investigación sea más eficiente al obtener los datos en formato y estructura familiarizado con sus conocimientos.

Objetivos

Objetivo general:

Diseñar e implementar una arquitectura de software middleware para gestión y agregación geoespacial de conjuntos de datos heterogéneos provenientes de estaciones climatológicas.

Objetivos específicos computacionales:

- Diseñar una arquitectura de software middleware para la integración de datos
- Desarrollar la arquitectura de software
- Desarrollar interfaces de programación de aplicaciones (API) para el consumo de datos.
- Establecer una metodología para la integración de nuevas fuentes de datos.
- Realizar el caso de estudio relacionado a la estación en Querétaro
- Realizar el caso de estudio relacionado al conjunto de datos GHCN

Para cada uno de los casos de estudio se plantean los siguientes objetivos:

- Analizar los datos utilizados en el caso de estudio.
- Integrar el conjunto de datos a la arquitectura.
- Desarrollar clientes que consuman los datos integrados

Objetivos específicos meteorológicos:

- Almacenar los datos generados por la estación EddyPro localizada en Querétaro.
- Desarrollar un cliente que permita descargar los datos de la estación de Querétaro en formato CSV y JSON.
- Almacenar los datos provenientes de los países México, Canadá y Estados Unidos en el conjunto de datos GHCN.
- Desarrollar un cliente que presente los datos de GHCN en una distribución espacial y permita realizar operaciones analíticas básicas en los datos.

Hipótesis

Estandarizar la integración de datos a través de la propuesta de una arquitectura de bien definida, así como sus lineamientos establecidos, de tal manera que al integrar una nuevo conjunto de datos se pueda seguir una metodología y pueda ser comprendido por los interesados, sean expertos en ciencias computacionales o no.

Capítulo 2.

Marco teórico

Conceptos computacionales

En el campo de computación, el problema descrito por esta investigación tiene una gran relación al concepto de integración de datos. Doan (2012) menciona que el objetivo de un sistema de integración de datos es ofrecer un acceso uniforme a un conjunto de fuentes de datos autónomas y heterogéneas. Convey (2001) menciona que los sistemas de integración de datos harmonizan datos provenientes de múltiples fuentes en un sola representación coherente. Las fuentes no son necesariamente bases de datos, pueden ser sistemas legados (sistema viejos y obsoletos que son difíciles de migrar a tecnología moderna) o archivos estructurados/no estructurados con interfaces diferentes.

Con la integración de datos los usuarios se pueden enfocar en especificar lo *que* desean, y no *cómo* obtener lo que desean. En lugar de buscar fuentes relevantes, interactuar con cada una y combinar datos de diferente origen, un usuario puede realizar consultas de una manera unificada. Además la integración de datos apoya las aplicaciones de soporte de decisiones como la minería de datos y el procesamiento analitico en linea.

Con base en uno de los objetivos de esta investigación se desea diseñar una arquitectura de software, definida por la IEEE (2011) como: “los conceptos o propiedades fundamentales de un sistema en su entorno representados en sus elementos, relaciones y en los principios de su diseño y evolución”. Existe una gran variedad de posibles arquitecturas para la integración de datos pero hablando en general, la mayoría de los sistemas se encuentran dentro del espectro entre *warehousing* e integración virtual. En el lado de *warehousing*, los datos de fuentes individuales son cargados y materializados a una base de datos física, llamada *data warehouse* definida por Inmon (2005) como “una recopilación de datos orientada a temas, integrada, no volátil y variante del tiempo en apoyo de las decisiones de la administración”. En la integración virtual, los datos son mantenidos en las fuentes y son accedidos según sea necesario al momento de consulta.

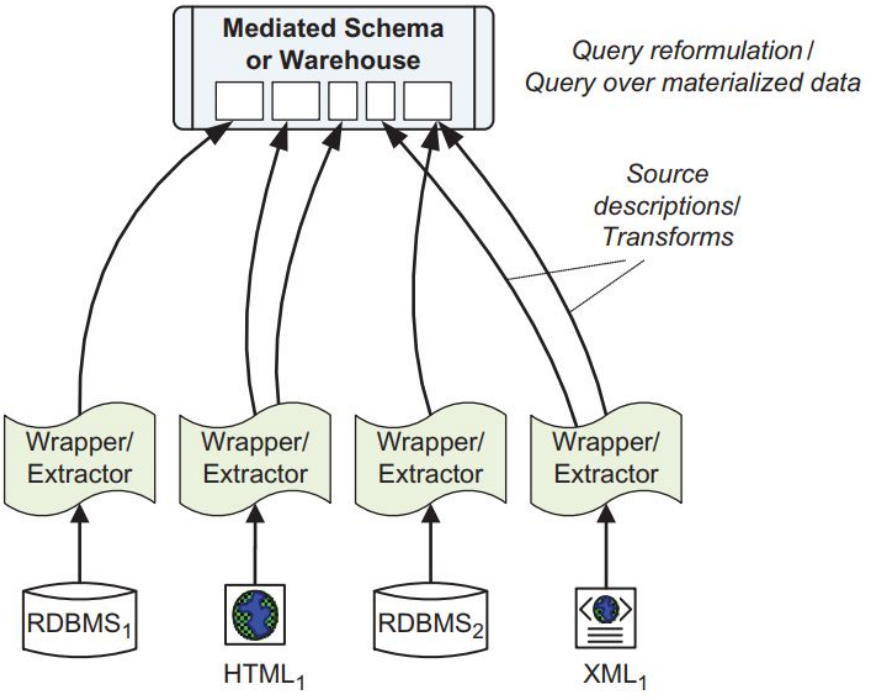


Figura 1. Arquitectura básica de un sistema de integración de datos de propósito general. Tomado de Doan (2012)

La Figura 1 muestra los componentes lógicos para ambos tipos de sistema de integración de datos. En la parte inferior de la figura se encuentran las fuentes de datos. Estas fuentes de datos pueden variar en dimensiones, como su modelo de datos o la clase de consultas que soportan. Arriba de las fuentes de datos se encuentran los programas cuyo rol es el de comunicarse con las fuentes de datos.

En la integración virtual, estos programas de software son llamados *wrappers* o envolturas, su rol es el de enviar las consultas a la fuente de datos, recibir respuestas y posiblemente aplicar una transformación básica en las respuesta. El usuario interactúa con el sistema de integración de datos a través de un solo esquema, llamado esquema medio que contiene sólo los aspectos de dominios que son relevantes para la aplicación. En el enfoque virtual el esquema medio no

almacena datos sino que funciona como un esquema lógico utilizado para plantear consultas.

La clave para consultar una aplicación de integración de datos son las descripciones de fuentes. Estas descripciones especifican las propiedades de las fuentes que el sistema necesita conocer para utilizar los datos. El componente principal de las descripciones de fuentes es el mapeo semántico que especifica cómo los atributos de las fuentes corresponden con los atributos del esquema medio.

En el enfoque de *data warehouse*, el usuario plantea consultas al esquema de *data warehouse* en vez del esquema medio. Además de ser un esquema que contiene los atributos necesarios de las fuentes, el esquema de *data warehouse* también es un esquema físico con un instancia de base de datos. A diferencia de *wrappers*, el sistema incluye herramientas de extracción-transformación-carga (ETL por sus siglas en inglés) que extraen datos de las fuentes periódicamente y los cargan al *data warehouse*.

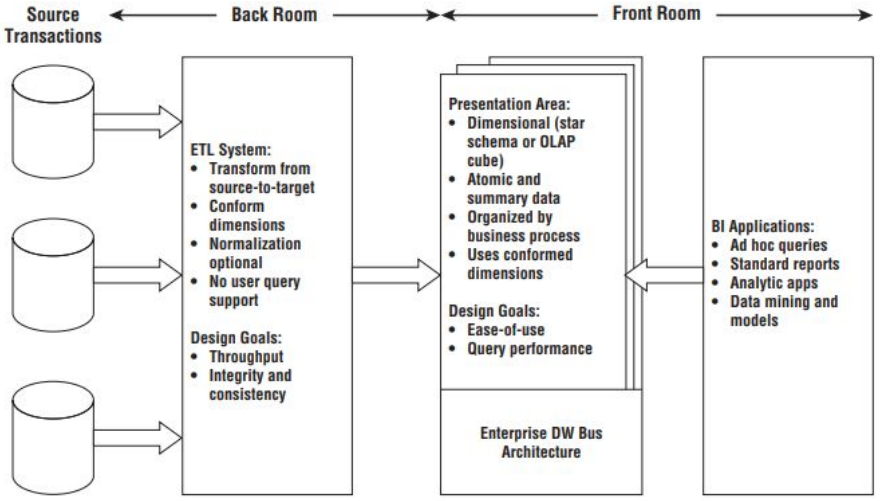


Figura 2. Elemento principales de la arquitectura de Kimball. Tomado de Kimball y Ross (2011)

La Extracción, Transformación y Carga (ETL por sus siglas en inglés) es un proceso en el uso de base de datos y específicamente en data warehouse. Como se puede apreciar en la Figura 2, el sistema ETL forma parte de la arquitectura de Data Warehouse donde se realizan los siguientes procesos descritos por Kimball (2013):

- Extracción es el primer paso en el proceso de obtención de datos. La extracción se refiere a la lectura y entendimiento de los datos fuentes y al proceso de copiar los datos necesarios al sistema ETL para su posterior manipulación.
- Después de extraer los datos, hay numerosas transformaciones potenciales, como la limpieza de datos (corrección de errores ortográficos, resolución de conflictos de dominio, tratar con elementos perdidos o convertirlos en formatos estándar), combinar datos de múltiples fuentes y eliminar datos duplicados.

- El último paso de proceso ETL es el estructuramiento y carga física de los datos en la área de presentación.

El proceso de ETL presentaba problemas en el campo de integración de datos casi en tiempo real. Estos problemas son:

- La repetición de extracción y transformación de datos que no tienden a cambiar para cada ventana de carga de datos.
- Sin soporte de *Plug-and-play* (Capacidad de agregar nuevas fuentes de datos al momento de ejecución)

Con el motivo de solucionar los problemas que presentaba el proceso, ETL Naeem, Dobbie y Webber (2008) proponen una arquitectura de integración de datos casi en tiempo real basada en eventos. En la integración de datos casi en tiempo real, la extracción y transformación de datos que no tienden a cambiar cuando los datos son extraídos y transformados a un nivel transaccional, no es factible extraer los datos por cada transacción. La falta de soporte de plug-and-play conduce a una falta de flexibilidad en el reemplazo de componentes en la arquitectura. Para corregir esto se propone una solución estándar basado en eventos para la integración de datos.

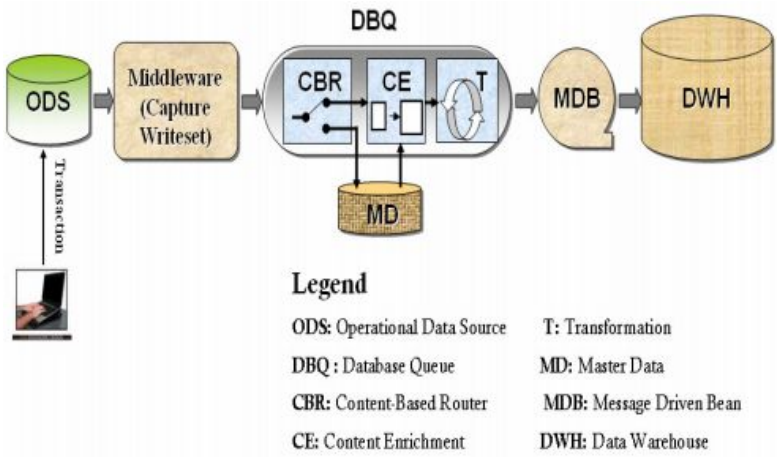


Figura 3. Arquitectura de integración de datos casi en tiempo real basada en eventos. Tomado de Naeem et al. (2008)

El middleware es la capa de software instalada sobre cada base de datos para capturar los cambios en la forma de un conjunto de escritura. La cola de base de datos (DBQ) es una cola orientada a mensajes que almacena los mensajes en tablas de base datos. El enrutador basado en contenido (CBR) analiza el mensaje en base de sus contenidos y los transmite al canal apropiado, los datos que no tienden a cambios son dirigidos a un repositorio mientras que los datos

transaccionales son propagados al proceso de enriquecimiento. El enriquecimiento de contenido (CE) es una forma especial de traducción de datos donde información adicional es inyectada al mensaje actual. Un bean impulsado por mensajes (MDB) examina continuamente la cola de base de datos, tan pronto como las actualizaciones son transformadas al formato requerido se extraen y cargar al data warehouse.

Otra arquitectura creada con el motivo de mejorar el proceso ETL fue la propuesta por Zhu (2004), una arquitectura basada en servicios (Figura 4) donde los proveedores de servicios publican sus fuentes de datos como servicios de accesos a datos (un tipo de servicio web que permite la provisión de un servicio intensivo de datos) que podría ser descubierto, enlazado en el momento en que son necesarios y desconectados después de su uso. La arquitectura propuesta está basada en la idea de Software como servicio y enlaces dinámicos.

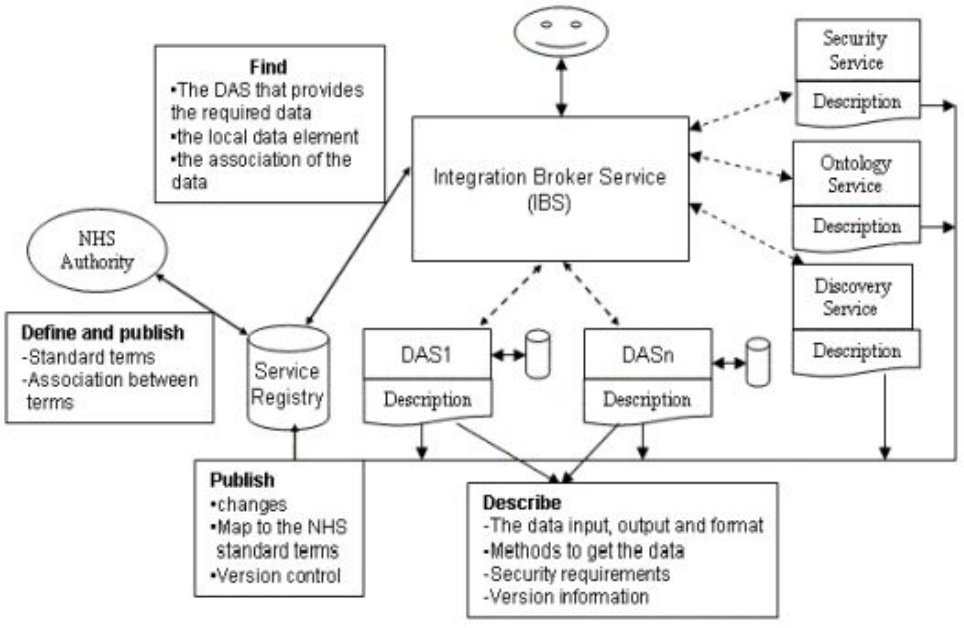


Figura 4.Arquitectura de integración de datos basada en servicios. Tomado de Zhu et al (2004)

Se asume que todos los elementos: contenidos de negocio (vocabulario), constante (medición, código postal, etc) y tipos de datos son definidos o descritos con un enfoque ontológico por la autoridad de la fuente. Los proveedores de servicio (incluyendo los proveedores de servicios de datos) implementan los servicios y los describen utilizando lenguajes de descripción de servicios como *Web Services Description Language* (WSDL) o *DARPA Agent Markup Language for Services* DAML-S. Todos los servicios pueden ser implementados utilizando diferentes lenguajes en diferentes plataformas. Los proveedores de servicios publican el archivo de descripción de servicio en el registro de servicios, como Descripción Universal, Descubrimiento e Integración (UDDI por sus siglas en inglés) un estándar basado en XML para describir, publicar y encontrar servicios web. Se mapean los elementos de datos locales a los términos estándar definidos por la autoridad.

Los servicios son localizados utilizando el registro de servicios y la implementación del servicio es invocada vía *Simple Object Access Protocol* (SOAP), un protocolo para el intercambia de información estructurada en la implementación de servicios web. El servicio de descubrimiento es utilizado para descubrir y enlazar implementaciones de servicios al momento de ejecución. El servicio de ontología es utilizado para proveer transformaciones semánticas para los datos. El servicio de seguridad es utilizando para autentificar al usuario y controlar los datos que puede acceder el usuario. El servicio intermediario de integración (IBS) es un servicio compuesto que integra diferentes servicios de acceso a datos y servicios

funcionales como el servicio de descubrimiento, seguridad y de ontología con el motivo de proveer al usuario final una vista unificada de los datos.

Un servicio de acceso a datos (DAS) es una variación de un servicio (web) típico ya que es más enfocado a datos y diseñado para exponer datos como un servicio. Los proveedores de servicios de datos deben describir la entrada de datos, la salida y formato, los métodos para adquirir los datos, los requerimientos de seguridad para utilizar el servicio y la versión del servicio. Cuando un proveedor de servicio de datos pública un archivo de descripción de DAS en el registro, pública los metadatos y ontología de DAS. Para el manejo de cambios, los proveedores de DAS pueden proveer a los usuarios diferentes versiones de DAS. No existe un esquema integrado; los datos son integrados sobre la marcha.

Los enfoques de integración de datos mencionados previamente se encuentran enfocados en casos específicos, sin embargo con el surgimiento de problemas recientes de *Big Data*, Dajda (2015) propone una arquitectura para dar respuesta a la necesidad de este nuevo enfoque que sea dinámico y escalable pero a su vez ligero. Dajda (2015) propuso una arquitectura basada en *plugins* para la integración que pueda manejar la distribución, descentralización y proveer extensibilidad y soporte para la reutilización de código.

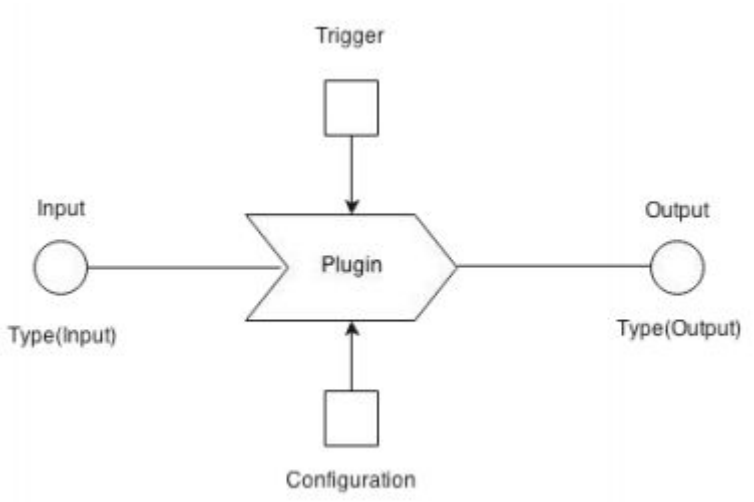


Figura 5. Arquitectura de *plugin*. Tomado de Dajda y Dobrowolski (2015)

El módulo básico de la arquitectura es llamado *plugin* (Figura 5). Al aplicar el concepto de *plugin*, la arquitectura llega a ser extendible y puede adaptarse a necesidades específicas al ensamblar flujos de datos con el conjunto disponibles de *plugins.* Un *plugin* tiene por lo menos una entrada y una salida que su habilidad de procesamiento de datos. Los *plugins* pueden ser combinados para formar un flujo (o flujos) para realizar un objetivo principal del sistema. La combinación de los *plugins* toma en consideración su compatibilidad.

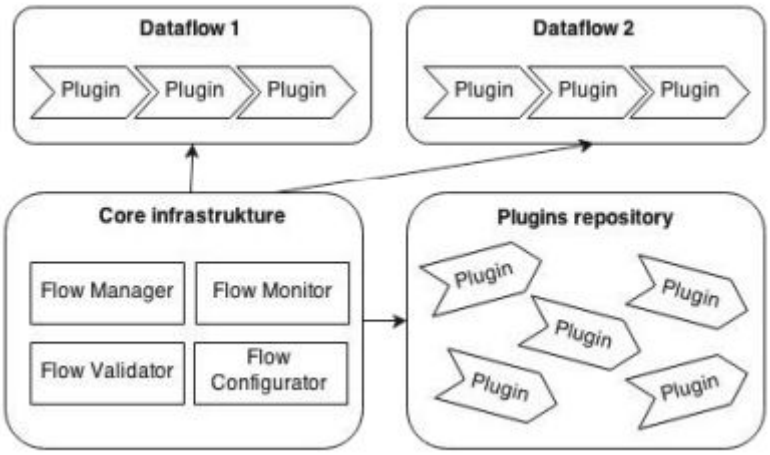


Figura 6. Arquitectura basada *plugins*. Tomado de Dajda y Dobrowolski (2015)

En la Figura 6 se presenta la arquitectura general del sistema. Un *plugin* es descrito por una interfaz que asume el formato de los datos de entrada o salida. Los *plugins* son clasificados en tres categorías:

- Fuente:

  Lectura de contenido de internet u otras fuentes con alguna filtración y entrada preparada para *plugins* de procesamiento.

- Final

  Cerrado del procesamiento con el almacenamiento o presentación de la información.

- Analítico

  Realizan un análisis de acuerdo con algoritmos complicados arbitrarios.

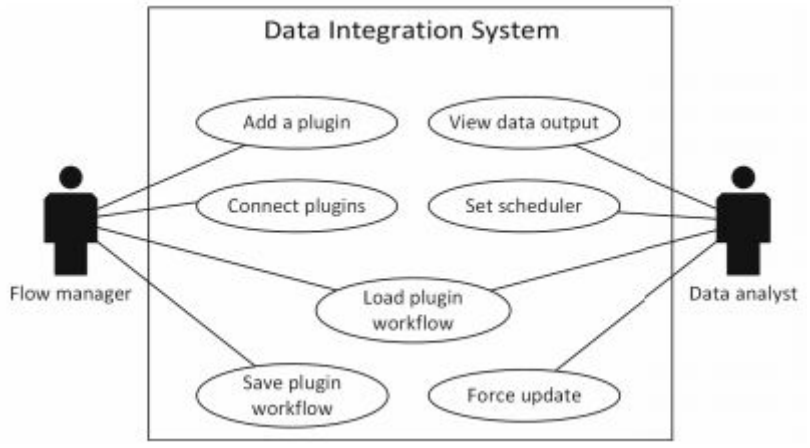


Figura 7. Perspectivas del uso del sistema. Tomado de Dajda y Dobrowolski (2015)

Desde el punto de vista humano (Figura 7) existen dos perspectivas del sistema: Administrador de flujo y Analista de datos. El analista de datos prepara un escenario para la integración de datos. El escenario es descrito de manera más o menos formal y describe las fuentes, el procesamiento aplicado y el objetivo de la tarea de integración propuesta. La descripción le proporciona al administrador de flujos con toda la información necesaria para orquestar el flujo de datos en el sistema preparando los *plugins* necesarios y conectandolos.

Con la investigación de las arquitecturas para la integración de datos se conoce una vista general del estado de arte y cuales son los enfoques utilizados. En el caso de esta investigación se decidió basarse principalmente en la arquitectura basada en *plugins* debido a su flexibilidad; sin embargo se realizó con algunas modificaciones para adaptarse al problema. A continuación se definirán los conceptos meteorológicos relevantes en esta investigación debido a los datos que los datos que se estarán utilizando pertenecen a esta disciplina.

Conceptos meteorológicos

Los conjuntos de datos utilizados en este proyecto provienen del campo de metereología definida por Spellman (2012) cómo: “la ciencia relacionada con la atmósfera y sus fenómenos; un meteorólogo observa la temperatura, densidad, vientos, nubes, precipitación y otras características de la atmósfera para dar cuenta de su estructura observada y evaluación en términos de las influencias externas y las leyes básicas de la física”. En este campo los datos generados pueden provenir de distintos tipos de fuentes:

- Estaciones

  Este tipo de datos son observaciones terrestres o de superficie obtenidas de instrumentos localizados en un punto espacial. Usualmente incluyen temperatura, punto de rocío, humedad relativa, precipitación, velocidad y dirección de viento, visibilidad, presión atmosférica y tipos de fenómenos hidrometeorológicos como granizo o niebla.

- Satélites

  Albergan instrumentos de sensoria remota colocados en órbita. Existen dos tipos de satélites: geoestacionario y órbita polar. Un satélite geoestacionario orbita la tierra arriba del ecuador en altitudes de 35,880 km. Una satelite de orbita polar circula la tierra a una altitud típica de 850 km en una trayectoria de norte a sur (o viceversa), pasando sobre los polos en su viaje continuo. Las observaciones obtenidas por este tipo de fuentes corresponden a múltiples puntos espaciales observados.

- Radar

  Término derivado del acrónimo en inglés Detección y Rango por Radio, es un instrumento para la detección de objetos que utiliza ondas de radio para determinar el rango, altitud, dirección de movimiento y velocidad de los objetos.

- Modelo

  Estos son procedimientos o algoritmos diseñados para generar nuevos datos a partir de una entrada. Algunos de estos modelos pueden ser utilizados para la predicción climática o modelos globales climáticos.

- Globo de clima

  Los datos climáticos de la atmósfera, empezando tres metros sobre la superficie de la Tierra son considerados datos de globos climáticos o datos de aire superior. Estos datos se obtienen a partir de radiosondas, que son emitidas por instrumentos atados a globos que se lanzan desde el suelo, ascienden a través de la troposfera hacia la estratosfera y vuelven a transmitir a una estación receptora en el suelo.

- Datos marinos/oceánicos

  Datos meteorológicos provenientes de naves en el mar, boyas amarradas y a la deriva, estaciones costeras y plataformas.

- Paleoclimatología

  Datos derivados de fuentes naturales como anillos de árboles, núcleos de hielo, corales y sedimentos de océanos y lagos.

Es importante conocer los distintos tipos de fuentes de datos para identificar el problema de heterogeneidad presentado en este campo ya que los diferentes tipos

de fuentes de datos generan datos que un experto requeriría unificar para realizar su análisis y posteriormente obtener la información de interés. A continuación se presentan los conceptos relacionados a los conjuntos de datos utilizados en esta investigación para conocer cómo son generados y accesados.

Conceptos relacionados con datos

En este proyecto de investigación se trabajó con una estación localizada en Bernal, Querétaro (Figura 8), en las coordenadas 20.717 y -99.941 de latitud y longitud respectivamente, perteneciente a la Universidad Autónoma de Querétaro utilizado en el proyecto “Heterogeneidad y escalamiento de los flujos de superficie de agua y energía, suelo-vegetación-atmósfera en sistemas climáticos regionales. Altiplano de México”. Esta estación es uno de los productos de la empresa LI-COR Biosciences que diseña, manufactura y vende instrumentos para la investigación biológica y ambiental.

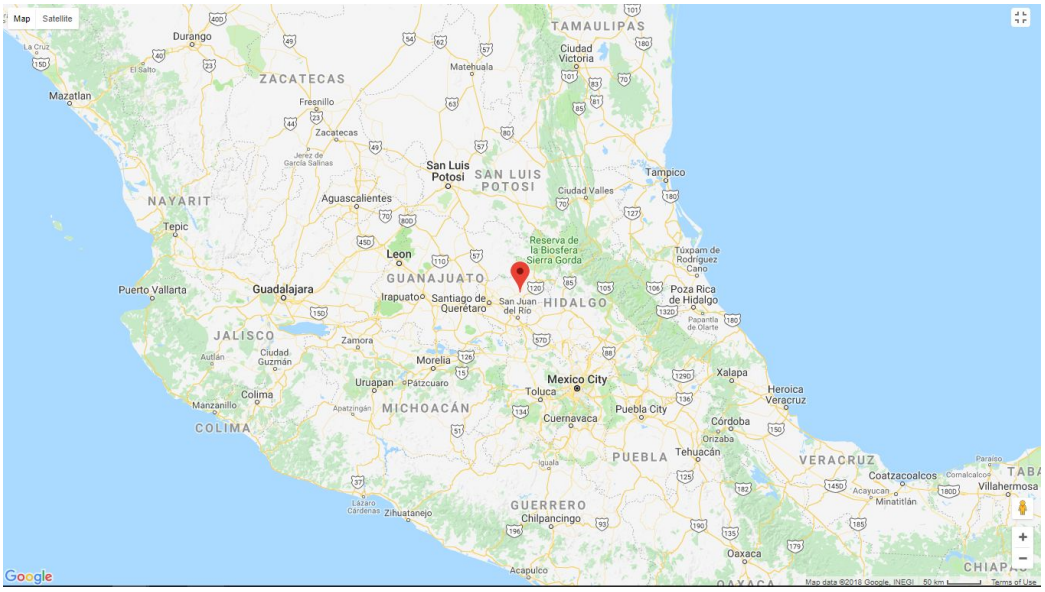


Figura 8. Localización de estación en Querétaro.

La estación en particular es un analizador de gas CO_2/H_2O LI-7500RS definida en el respectivo manual como “un analizador infrarrojo de gas de alta velocidad y precisión que mide de manera precisa las densidades del dióxido de carbono y el vapor de agua en las estructuras de aire turbulentas”. El listado de las variables observadas por la estación son:

- Tiempo
- Fecha
- CO_2 (mmol/m^3, absorción, o µmol/mol)
- H_2O (mmol/m^3, absorción, o µmol/mol)
- Punto de rocío (°C)
- Presión (kPa)
- Temperatura
- Voltaje de enfriador (V)
- Valor de diagnóstico
- Entrada auxiliar 1 (e.g voltaje sonico U)
- Entrada auxiliar 2 (e.g voltaje sonico V)
- Entrada auxiliar 3 (e.g voltaje sonico W)
- Entrada auxiliar 4 (e.g voltaje de temperatura sonico)

Los datos generados por la estación son de formato GHG, un formato de datos crudos que consiste en un archivador que contiene el archivos de datos (extensión “.data”) y archivo de metadatos (extensión “.metadata”). El archivo de datos es una tabla ASCII con encabezado que tiene un número de renglones y columnas,

separados por el carácter tabulador.

El dispositivo cuenta con una Unidad de Interfaz del Analizador que permite la colección de conjuntos de datos completos. Esta unidad utilizan el software EddyPro para procesar los datos crudos observados. El software EddyPro es una aplicación que procesa los datos para calcular flujos de vapor de agua (evapotranspiración), dióxido de carbono, metano, otros gases traza y energía con el método Eddy Covariance. Otro software utilizado por esta unidad es FluxSuite que se encarga de poner en línea los resultados del sistema Eddy Covariance.

Otro de los conjuntos utilizados es la Red Global de Climatología Histórica (GHCN por sus siglas en inglés) es una base de datos de la Administración Nacional Oceánica y Atmosférica (NOAA por sus siglas en inglés), esta base de datos es integrada por resúmenes climáticos de estaciones terrestres a lo largo del mundo que han sido sujetas a reseñas de calidad. Algunos de los datos tienen más de 175 años mientras otras menores a una hora. Existen dos tipos de conjuntos GHCN uno con datos diarios y otro con promedios mensuales. El acceso a los datos es utilizado por medio de un servidor FTP o bien por una herramienta para la búsqueda de datos.

Ambos conjuntos de datos provienen de estaciones lo que significa que los datos son puntuales (representan observaciones de puntos georeferenciados). Mientras que el conjunto de datos de la estación de Querétaro representa un solo punto en el mapa, el conjunto de datos GHCN se compone de distintas estaciones dispersas alrededor del mundo. En la Figura 9 se muestran los círculos que indican la localización de las estaciones agrupadas por la cercanía de otras estaciones, el

número dentros de estos círculos simboliza la cantidad de estaciones dentro de este grupo. Además en la Figura 9, se aprecia la dispersión de estación y como existen puntos donde no se encuentran observaciones para estimar los valores, en los puntos desconocidos se utiliza el concepto de interpolación de datos.

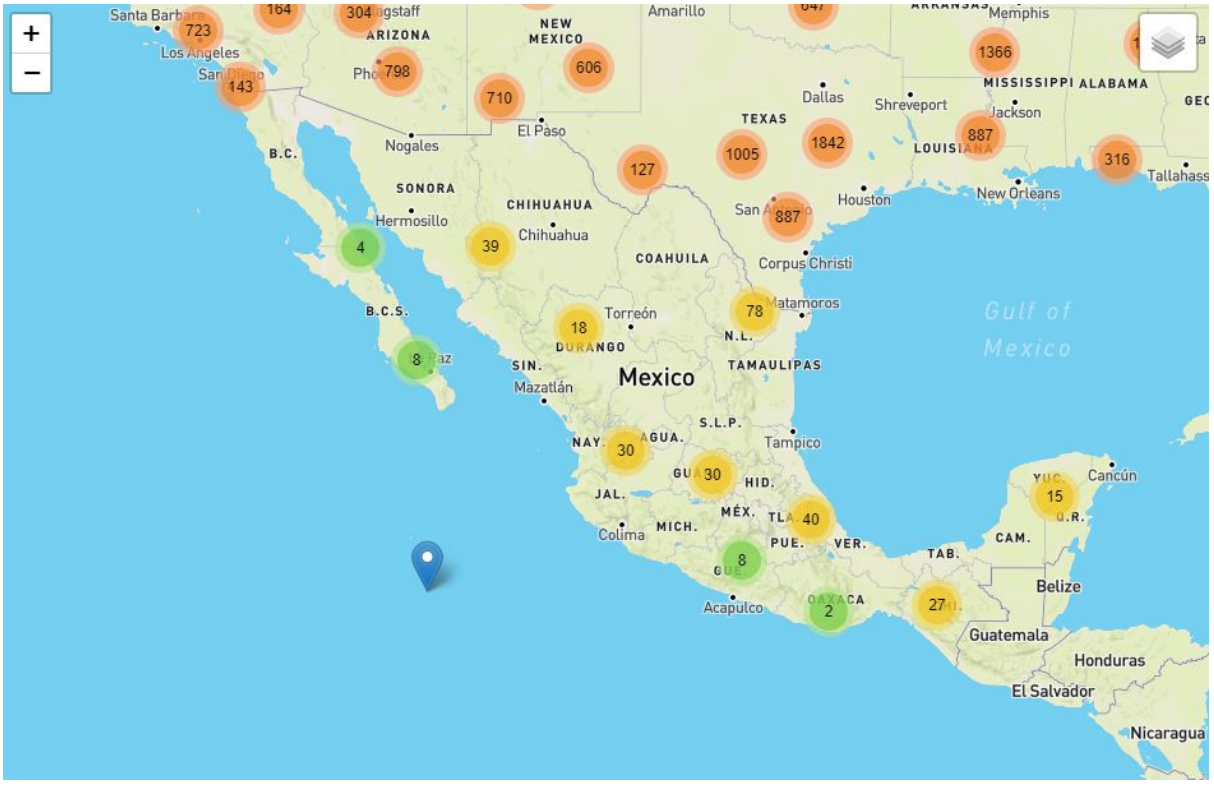


Figura 9. Estaciones de GHCN agrupadas dispersas.

La interpolación de datos es definida por Zhan (2012) como: “un método de construir nuevos puntos de datos dentro del rango de un conjunto discreto de puntos conocidos”. Muchos de los métodos de interpolación fueron desarrollados para predecir el valor de fenómenos espaciales en lugares no muestreados. Mitas (1999) describe algunos de estos métodos:

- Enfoque vecinal local

  Los métodos locales están basados en la suposición que cada punto influye al resultado de la superficie solo hasta una cierta distancia finita. Dentro de este enfoque se encuentran los métodos:

  - Interpolación ponderada por distancia inversa (IDW)

    Está basado en la suposición que el valor de un punto muestreado puede ser aproximado como un promedio ponderado de valores, en puntos dentro de una cierta distancia o un número determinado de puntos cercanos. El método a menudo no reproduce la forma local implícita por los datos y produce extremos locales.

  - Vecino Natural

    Este utiliza un promedio ponderado de los datos locales basado en el concepto de coordenadas de vecino natural, derivado de los polígonos de Thiessen para bivariado y los poliedros de Thiessen para trivariado. El valor en una locación no muestreada es calculado como un promedio ponderado de los valores de los vecinos cercanos con ponderación dependiendo de áreas o volúmenes en vez de distancias.

  - Interpolación basada en una red triangular irregular (TIN)

    Este utiliza un mosaico triangular de un punto dado para derivar una función bivariada para cada triángulo, que después es usada para estimar los valores en las locaciones no muestreadas.

  - Métodos basados en rectángulos

    Estos son análogos a TIN e involucra el ajuste de funciones

polinómicas combinadas para rectángulos regulares o irregulares.

- Enfoque geostático

  El método *Kriging* está basado en el concepto de funciones aleatorias: la superficie o volumen son asumidos en ser una realización de una función aleatoria con una cierta covarianza espacial. En general, *Kriging* predice los valores en puntos y bloques en un espacio dimensional *d,* y permite la incorporación de anisotropía. Las principales fortalezas de *Kriging* se encuentran en la calidad estadística de sus predicciones y la habilidad para predecir la distribución espacial de la incertidumbre.

- El enfoque variacional

  Está basado en la suposición que la función de interpolación debe pasar a través (o cerca de) los puntos y, al mismo tiempo, debe ser los más fluido posible.

Mientras que el método seleccionado para la interpolación de datos puede variar el valor obtenido, el propósito de la interpolación de datos es obtener los puntos desconocidos. La interpolación de datos en los datos puntuales puede ser utilizada para generar datos en malla donde los puntos se encuentran en distancias uniformes donde la distancia es conocida como la resolución. Los datos en malla usualmente son la estructura de datos que generan los satélites. A través de estos datos en malla se pueden realizar representaciones visuales para facilitar el análisis como los mapas de calor.

Capítulo 3.

Metodología

Posterior a la investigación del estado del arte, se realizó un análisis comparativo de las arquitecturas de integración de datos. El análisis consistió en identificar las ventajas y desventajas de cada una de las arquitecturas, con el objetivo de seleccionar una arquitectura para basar el diseño de la arquitectura propuesta. Cabe mencionar que las ventajas y desventajas fueron identificadas en el contexto del problema.

Con los resultados obtenidos del análisis se diseñó una arquitectura de software con base al Modelo “4+1” (Figura 11) propuesto por Kruchten (1995) el cual describe como: “Un modelo para describir la arquitectura de sistemas de software basado en el uso de múltiples vistas concurrentes. El uso de múltiples vistas permite afrontar los intereses de los varios ‘*stakeholders*’ de la arquitectura y administrar por separado los requerimientos funcionales y no funcionales”. El “1” en “4+1” representa los escenarios o casos de uso para la validación e ilustración de la arquitectura, el “4” representa cuatro vistas principales:

- Vista lógica
- Vista de procesos
- Vista de desarrollo
- Vista física

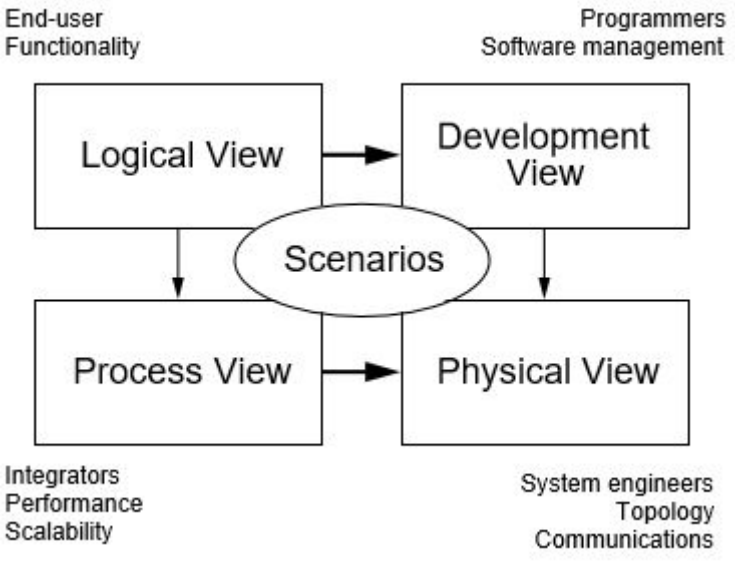


Figura 10. Esquema del modelo “4+1”. Tomado de Kruchten (1995)

Una vez que la arquitectura fue diseñada esta debe ser desarrollada en los lenguajes de programación que brindarán las funcionalidades necesarias para implementar la arquitectura en cuestión. Para lograr comprobar que la arquitectura funciona correctamente se plantearon dos casos de estudio, uno correspondiente a la estación de Querétaro y otro al conjunto de datos GHCN.

Un caso de estudio comprende la integración de una nueva fuente de datos, para realizar esto se debe iniciar con la adquisición de los requerimientos para conocer los requisitos que se deberán cumplir para considerar el caso de estudio como satisfactorio. Posteriormente se realizó un análisis a la fuente de datos para comprender los detalles de la misma, estos detalles pueden ser: el formato de origen de los datos, el instrumento que generó los datos, los procedimientos que se realizaron en los datos, entre otros. Se seleccionó un sistema gestor de base de datos (SGBD) que fue el que mejor se adecuó a los datos considerando los detalles obtenidos en el análisis de la fuente de datos.

Una vez que obtuvo la información previa se desarrolló un software que se encarga de la conversión del formato de origen al formato adecuado para su almacenamiento en el SGDB seleccionado. Siguiendo las directrices de la arquitectura se integró el software desarrollado y una vez integrado a la arquitectura se realizó la conversión de los datos y se almacenó en el SGDB. Para finalizar el caso de estudio se desarrolló un software cliente para comprobar acceso a los datos previamente almacenados.

Capítulo 4.

Análisis de arquitecturas

Como primer paso en la obtención de resultados se realizó un análisis comparativo de las arquitecturas investigadas (Tabla 1). Con base en la comparación realizada se consideran los siguientes puntos:

- El proceso ETL y la arquitectura basada en eventos están enfocadas al *data warehouse*, lo que conlleva una solución muy amplia para lo especializado del problema.
- La arquitectura basada en servicios provee una flexibilidad que se desea para la solución, ya que se estarán integrando nuevas fuentes de datos, la desventaja se encuentra en el ciclo de diseño que existe al querer integrar una nueva fuente de datos.
- En comparación con la arquitectura basada en servicios, la arquitectura basada en *plugins* es más ligera proveyendo la misma flexibilidad, la desventaja a considerar es el tiempo de desarrollo de los *plugins* comunes que permitirán crear los flujos de datos.

Para este proyecto se seleccionó la arquitectura basada en *plugins* debido a que se considera que la solución al problema debe ser lo más ligero posible para utilizar la menor cantidad de recursos.

Tabla 1. Análisis comparativo de arquitecturas

Arquitectura	Ventajas	Desventajas
ETL	- Utilizado en el enfoque *data warehouse* - Existe una gran cantidad de herramientas con esta arquitectura	- Es un proceso de *data warehouse* que no especifica los elementos de la arquitectura
Casi en tiempo real basada en eventos	- Mejora el proceso ETL - Permite el *plug-and-play* - No realiza la extracción y transformación de datos que no tienden a cambiar	- La arquitectura tiene el objetivo de utilizarse en *data warehouse*
Basada en servicios	- Uso de servicios web - Flexibilidad	- Requiere de una etapa de diseño de servicio al integrar una nueva fuente de datos
Basada en *plugins*	- Flexible - Manejo de problemas de *Big Data* - Ligero y dinámico	- Requiere de un mayor tiempo de desarrollo al inicio para crear los *plugins* más comunes.

Descripción de la arquitectura

Con base en lo establecido previamente, se utilizó como fundamento la arquitectura basada en plugins a la cual se le realizaron las modificaciones correspondientes para su uso en la solución. En la arquitectura propuesta por Dajda (2015) se mantiene un repositorio de plugins, cada uno con la función de cumplir con los procedimientos comunes en el proceso de integración de datos como: limpieza de datos, convertir de un formato a otro, etc. La arquitectura cuenta con la capacidad de combinar los distintos plugins en una secuencia para crear un flujo de datos.

En el caso de arquitectura propuesta en este documento no se considerarán los flujos de datos, puesto que representan un mayor tiempo de desarrollo, en cambio se desarrollará un plugin por cada fuente de datos a integrar; de esta manera podemos decir que la fuente de datos ha sido “conectada”. Cabe mencionar que la posibilidad de añadir un flujo de datos está considerada como un ciclo de desarrollo posterior, una vez que se hayan integrado suficientes fuentes de datos para poder abstraer los procedimientos comunes.

La decisión de mantener un plugin por cada fuente de datos se debe a que las fuentes de datos provenientes del mismo origen usualmente sufren cambios en sus nuevas versiones; estos cambios pueden encontrarse en su estructura o formato. Con este enfoque de un plugin por fuente de datos podemos mantener integradas las diferentes versiones de las fuentes de datos. Además, este enfoque simplifica el desarrollo de plugins ya que se propone que la tarea de cada plugin sea el almacenamiento de los datos en un SGDB y que los procesos de aseguramiento de calidad de los datos sean realizados previamente a su integración. Los procesos de aseguramiento de calidad verifican las siguientes características de los datos: integridad, exactitud, pertinencia, accesibilidad, consistencia y confiabilidad.

Otro de los cambios en cuanto a la arquitectura propuesta por Dajda es que el resultado de la conversión de los datos no es provista al usuario a solicitud, sino que los datos son convertidos en un formato homogéneo; este formato es un SGBD, ya que el campo de metereología maneja conjuntos de datos de alto volumen lo que significa que si se solicita un conjunto de datos en un formato en particular esto

podrá tener un tiempo de espera muy largo en comparación al tiempo que toma convertir datos en un formato conocido.

Para definir la arquitectura se consideró definir los procesos que se llevarán a cabo para realizar la integración de los datos. Los procesos son: administración de los plugins, conversión de los datos, almacenamiento de los datos y obtención de los datos. Siguiendo lo propuesto por el modelo “4+1” se muestran las cuatro vistas de la arquitectura.

Vista de desarrollo

Esta vista se centra en la organización de módulos de software en el ambiente de desarrollo. El software es empaquetado en pequeñas partes (como librerías o subsistemas). Los subsistemas están organizados en una jerarquía de capas en donde cada capa provee de una interfaz bien definida para las capas superiores a ella. En la figura 11 se muestra la vista de desarrollo que se encuentra divida en tres capas principales: La Interfaz de Aplicaciones de Programación (API), Middleware de plugins y datos.

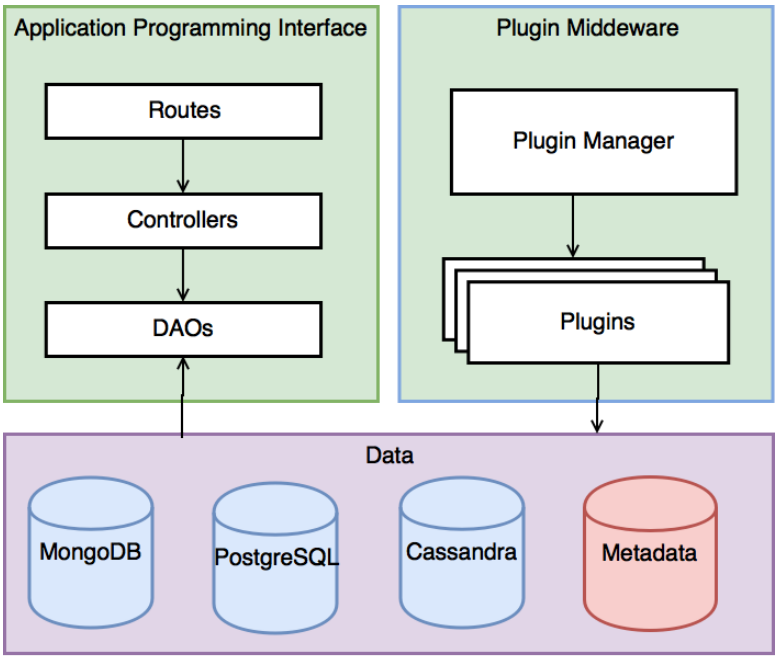


Figura 11. Vista de desarrollo.

El middleware de plugins se compone de un administrador de plugins (Plugin Manager) y los plugins. Esta capa realiza los procesos de administración de plugins, conversión y almacenamiento de datos. El administrador de plugins tiene el propósito de cargar y registrar los plugins al momento de ejecución del sistema; después de realizar esto se puede solicitar el plugin deseado y posteriormente ejecutarlo. Los plugins son los elementos de principales del sistema, se encargan de convertir los datos heterogéneos a un formato homogéneo y debe seguir las directrices establecidas.

Los datos se componen de múltiples bases de datos en distintos SGBDs y una base de datos con los metadatos. Los metadatos brindan la posibilidad de una etapa de descubrimiento para conocer los conjuntos de datos integrados y cómo fueron integrados. La Interfaz de Aplicaciones de Programación (API) se encarga del proceso de obtención de datos. La API se compone de las rutas de los métodos

(Tabla 2) que se ejecutarán para obtener los datos, estos métodos se encuentran definidos en los controladores donde se traduce la solicitud recibida a instrucciones para el Objeto de Acceso a Datos (DAO). El DAO es una patrón de diseño que desempeña el rol de intermediario entre el controlador y los SGBDs.

Tabla 2. Métodos del API

Ruta	Método	Parámetros	Regresa
/files/:db_id	GET	fields String Esta es una cadena que describe los campos de los metadatos que se desean obtener. Por defecto se obtienen todos los campos. Ejemplo: “name state size path” skip Integer El número de registros que se desea ignorar. El valor por defecto es 0. limit Integer El número de registros que desean obtener, en caso de que existen menos registros que el dado se obtendrán todos. El valor por defecto es 20.	Este método regresa los registros de los archivos relacionados con la base de datos especificada con el id en la ruta. Cada registro contiene los siguientes datos: name: string, state: string database: { name: string _id: UUID }, size: number, path: string
/plugin	GET	fields String Esta es una cadena que describe los campos de los metadatos que se desean obtener. Por defecto se obtienen todos los campos. Ejemplo: “name version description” skip Integer El número de registros que se desea ignorar. El valor por defecto es 0.	Este método regresa los registros de los plugins utilizados en la arquitectura. Cada registro contiene los siguientes datos: name: string, version: string, description: string, target_OS: string, target_DB: string, supported_files: { format: string },

		limit Integer El número de registros que desean obtener, en caso de que existen menos registros que el dado se obtendrán todos. El valor por defecto es 20.	dependencies: { name: string, type: string, }, help_file: number, vendor: string

/db/metadata	GET	fields String Esta es una cadena que describe los campos de los metadatos que se desean obtener. Por defecto se obtienen todos los campos.
		Ejemplo: “name created_at created_by”
		
		skip Integer El número de registros que se desea ignorar. El valor por defecto es 0.
		
		limit Integer El número de registros que desean obtener, en caso de que existen menos registros que el dado se obtendrán todos. El valor por defecto es 20.
			Este método regresa los registros de los metadatos de cada base de datos utilizada en la arquitectura.
			Cada registro contiene los siguientes datos:
			
			name: string,
			plugin: {
			name: string,
			version: string,
			description: string,
			target_OS: string,
			target_DB: string,
			supported_files:{
			format: string
			},
			dependencies: {
			name: string,
			type: string,
			},
			help_file: number,
			vendor: string
			},
			created_at: Date,
			created_by: string,
			updated_at: Date,
			db_schema: [{
			table: string,
			variables: [{
			name: string,
			description: string,
			type: string,
			size: number
			}]
			}]

/db/:id/metadata	GET	fields String Esta es una cadena que describe los campos de los metadatos que se desean obtener. Por defecto se obtienen todos los campos.
		Ejemplo: “name created_at created_by”
			Este método regresa los metadatos relacionados con la base de datos especificada con el id en la ruta.
			Los metadatos cuentan con los siguientes elementos:
			
			name: string,
			plugin: {
			name: string,
			version: string,
			description: string,
			target_OS: string,
			target_DB: string,
			supported_files:{
			format: string
			},
			dependencies: {
			name: string,
			type: string,
			},
			help_file: number,
			vendor: string
			},
			created_at: Date,
			created_by: string,
			updated_at: Date,
			db_schema: [{
			table: string,
			variables: [{
			name: string,
			description: string,
			type: string,
			size: number
			}]
			}]

/db/:id/data	GET	table String Este es el nombre de la tabla o colección que desea consultar. Este valor es requerido. fields String Esta es una cadena que describe los campos de los metadatos que	Este método regresa los datos relacionados con la base de datos especificada con el id en la ruta. Los datos contienen la estructura específica de la base de datos.

		se desean obtener. Por defecto se obtienen todos los campos. Ejemplo: “temp prec fecha” skip Integer El número de registros que se desea ignorar. El valor por defecto es 0. limit Integer El número de registros que desean obtener, en caso de que existen menos registros que el dado se obtendrán todos. El valor por defecto es 20. query String Este elemento debe ser una cadena convertible a JSON. Este elemento es un JSON que utiliza la sintaxis de consulta de MongoDB. El valor por defecto es { }. Ejemplo: “{ date: { $gte: “2016-08-18”, $lt: “2016-08-19” }}”	
/db/:id/export	POST	table String Este es el nombre de la tabla o colección que desea consultar. Este valor es requerido. fields String Esta es una cadena que describe los campos de los metadatos que se desean obtener. Por defecto se obtienen todos los campos. Ejemplo: “temp prec fecha” skip Integer El número de registros que se desea ignorar. El valor por defecto es 0.	Este método regresa el o los nombres de los archivos que se han creado en la exportación. La estructura es la siguiente: files: [string]

		limit Integer El número de registros que desean obtener, en caso de que existen menos registros que el dado se obtendrán todos. El valor por defecto es 20. query String Este elemento debe ser una cadena convertible a JSON. Este elemento es un JSON que utiliza la sintaxis de consulta de MongoDB. El valor por defecto es { }. Ejemplo: “{ date: { $gte: “2016-08-18”, $lt: “2016-08-19” }}” filetype string Esta cadena especifica el tipo de archivo en el que desea exportar los datos. El valor defecto es “csv”.	
/exports/:name	GET	Ninguno	Este método descarga el archivo especificado con el nombre establecido en variable name de la ruta.

Vista lógica

Esta vista muestra lo que el sistema debe proveer en términos de servicios para los usuarios y con esto apoyar a los requerimientos funcionales. El sistema se descompone en un conjunto de abstracciones claves, tomado del dominio del problema, en la forma de objetos o clases. En la figura 12 se muestra la vista lógica del middleware, dividido en tres secciones: Gestor de plugins (verde), interfaces de plugin (rojo) e implementación de plugin (azul).

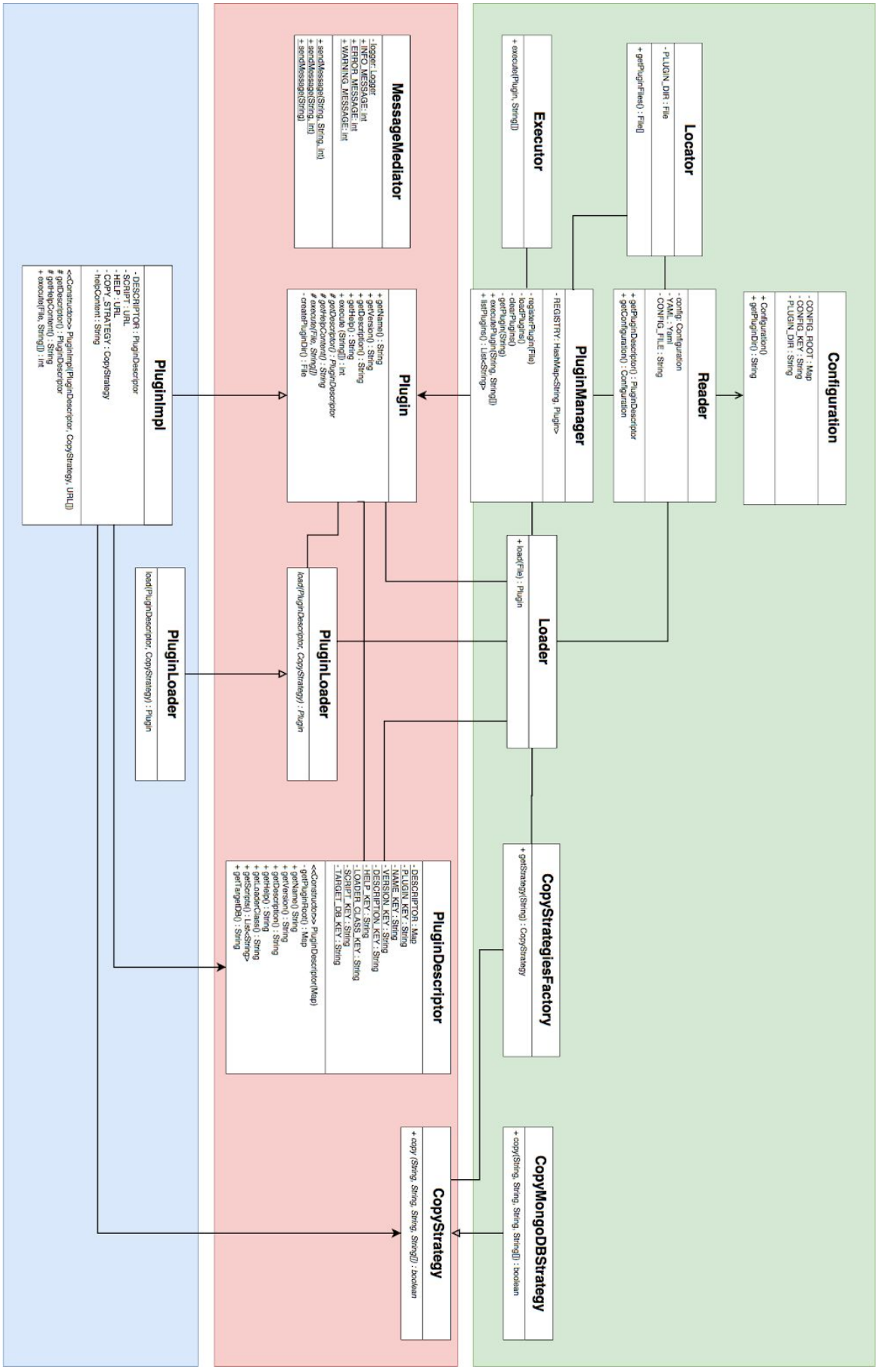


Figura 12. Vista lógica de la arquitectura del *middleware*

El gestor de plugins (Figura 13) se compone de las clases necesarias para la administración de los plugins. La clase principal de esta sección es el PluginManager que se encarga de inicializar y registrar los plugins con ayuda de las clases auxiliares:

- Executor: Ejecuta el plugin en un proceso independiente.
- Locator: Localiza todos los plugins.
- Reader: Lee el archivo de descripción del plugin.
- Loader: Inicializa el plugin a través del archivo de descripción.

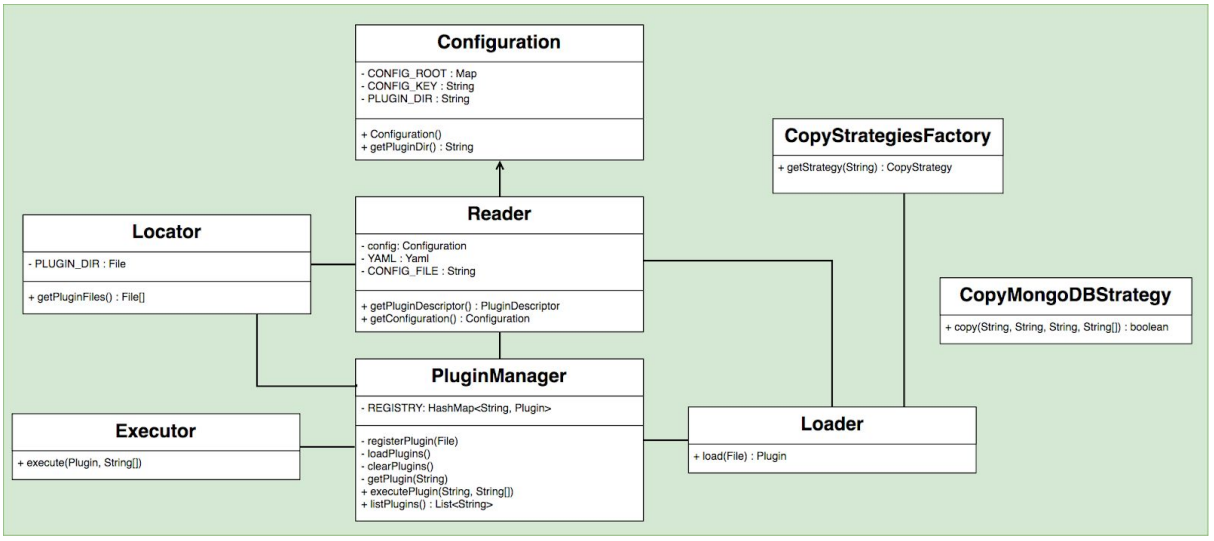


Figura 13. Vista lógica de gestor de *plugins*

Las interfaces de plugin (Figura 14), como su nombre lo indica, son en la interfaz entre el administrador y los plugins. Las interfaces son implementadas por los plugins, esto le permite al administrador conocer el comportamiento de los plugins. Las interfaces utilizadas son las siguientes:

- Plugin: Contiene el método a implementar que se ejecutará al momento de ejecutar el plugin.

- PluginLoader: Contiene el método a implementar que se ejecutará al momento de inicializar el plugin.
- PluginDescriptor: Es la representación en objeto del archivo de descripción del plugin.
- CopyStrategy: Utiliza el patrón de diseño de estrategia para encapsular el algoritmo de copiado de datos a un SGBD.

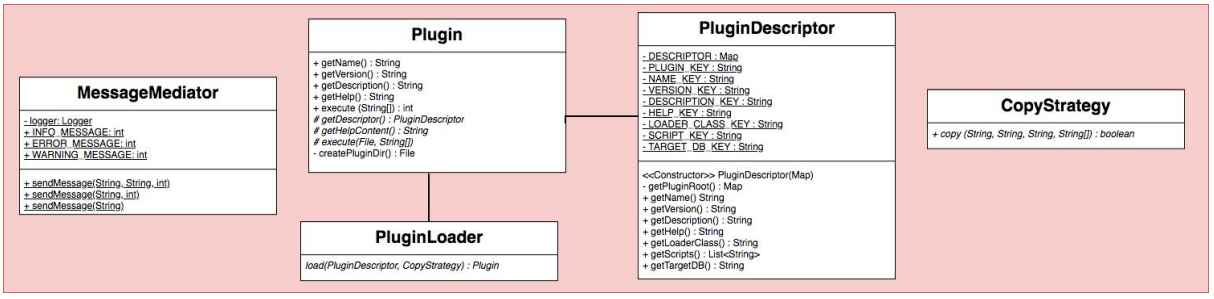


Figura 14. Vista lógica de las interfaces de *plugin*

La implementación del plugin (Figura 15) utiliza las interfaces para implementar la funcionalidad del plugin. El método de la interfaz PluginLoader es ejecutado al inicializar el plugin, esto no significa que el plugin se ejecutará, sino que será registrado en el PluginManager. El método de PluginLoader puede ser utilizado para preparar el entorno antes de ejecutar el plugin. El método de la interfaz Plugin es llamado al ejecutar el plugin, en este método se realizará la conversión de los datos y, con ayuda de CopyStrategy, se almacenarán los datos en el SGBD especificado en el archivo de descripción.

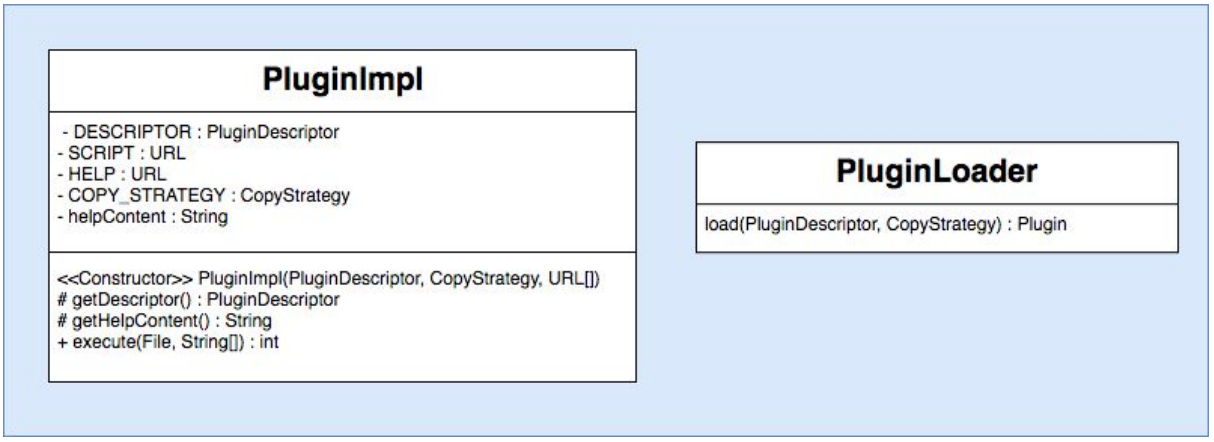


Figura 15. Vista lógica de la implementación del *plugin*

Vista de proceso

La vista de proceso toma en cuenta los requerimientos no funcionales, como el rendimiento y disponibilidad. Puede ser descrita como un conjunto de redes lógicas ejecutadas independientemente (llamados procesos), distribuidos en un conjunto de recursos de hardware. Un proceso es la agrupación de tareas que forman una unidad ejecutable.

Como ya se había mencionado previamente, los procesos principales que se llevan a cabo en la arquitectura son los siguientes: Almacenamiento de datos, administración de plugins y obtención de datos. En la figura 16 se presenta, con ayuda de un diagrama de flujo, el proceso de almacenamiento de datos donde interactúan el PluginManager, plugin, CopyStrategy y el SGBD. Para empezar a realizar el almacenamiento de los datos se necesita obtener el plugin correspondiente a la fuente de datos que se desea integrar, si el PluginManager no encuentra el plugin, este envía un mensaje de error y termina el proceso, en caso contrario al obtener el plugin se ejecuta el método execute() de la interfaz Plugin.

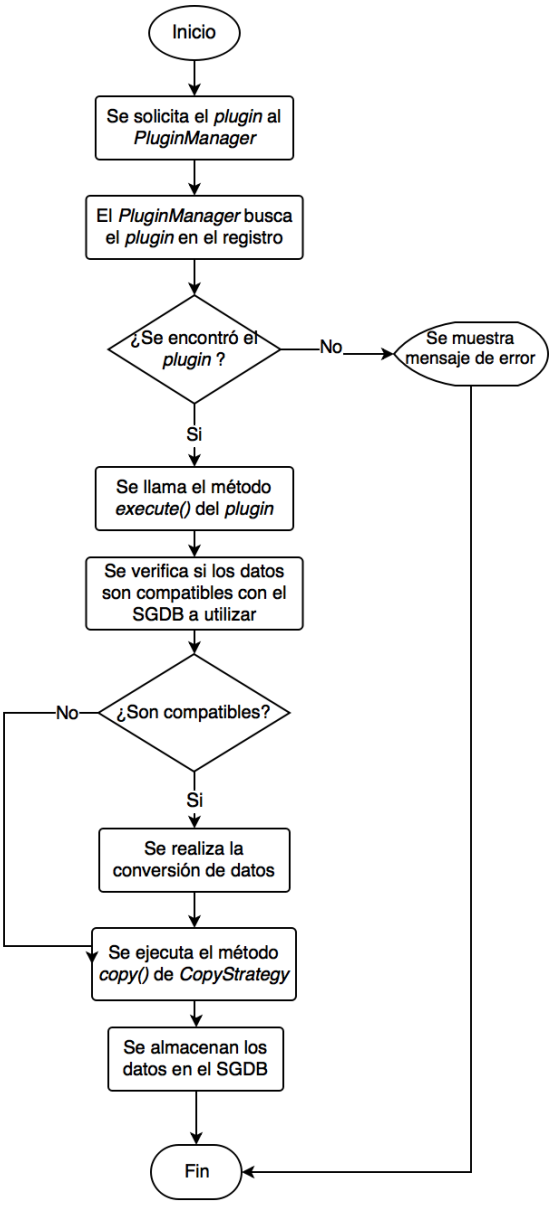


Figura 16. Diagrama de flujo del proceso “Almacenamiento de datos”

Antes de almacenarlos se debe verificar que los datos sean compatibles con el SGBDB a utilizar, esta compatibilidad se refiere a la capacidad de manejo del SGBD de los datos a almacenar, usualmente los formatos más compatibles son los separados por tabulador o coma pero existen formatos específicos de la fuente de datos como el formato GHG específico de la estación LICOR; el SGBD no cuenta con los mecanismos para almacenar este tipo de formatos. Cuando los datos no son compatibles debe realizarse una actividad previa al almacenamiento, donde los

datos sufren una reestructuración por parte del plugin para cumplir con la compatibilidad del SGBD. Una vez que los datos son compatibles, se utiliza la CopyStrategy del SGDB que encapsula el algoritmo para almacenar los datos en el gestor a través del método copy().

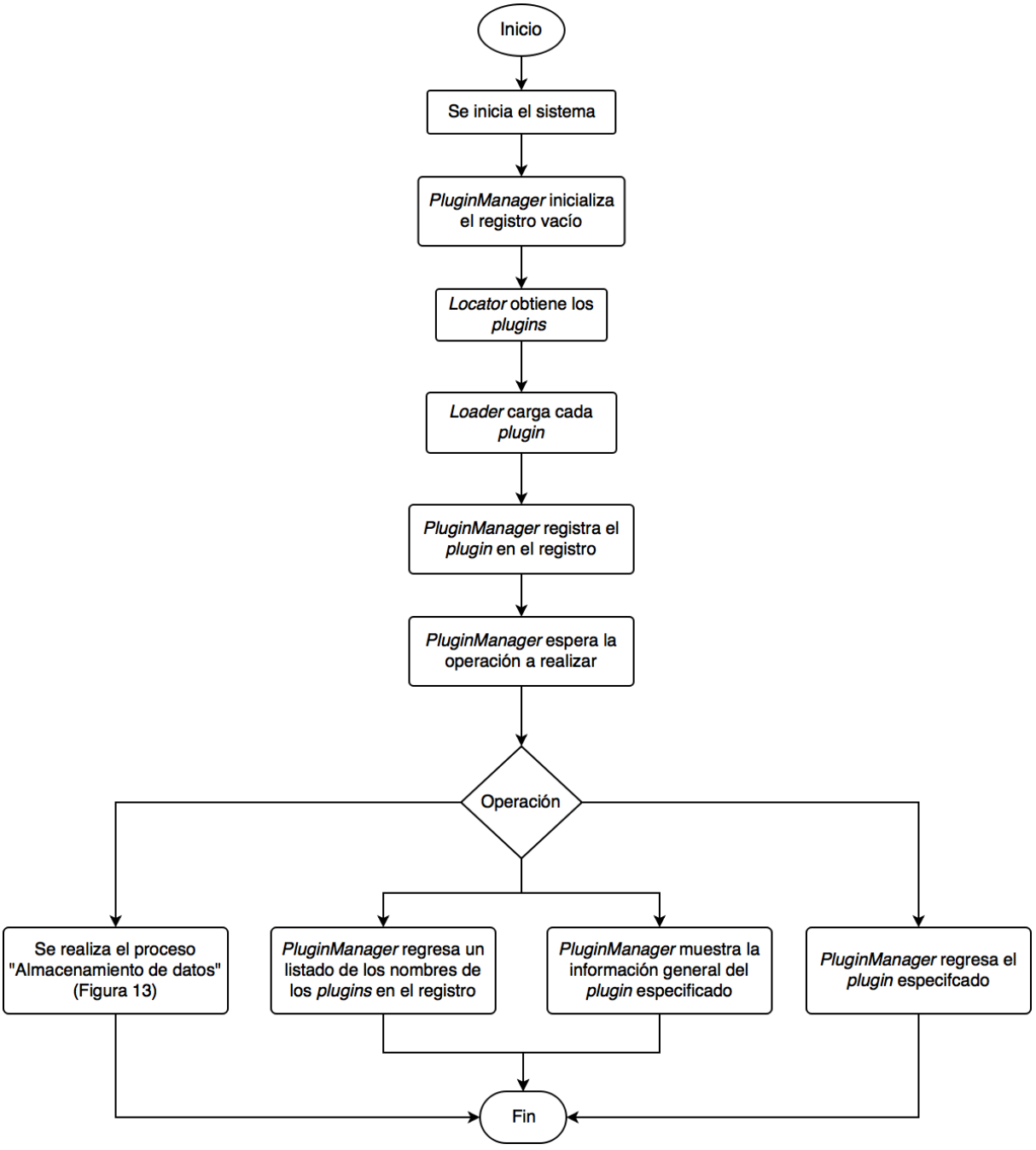


Figura 17 Diagrama de flujo del proceso “Administración de *plugins*”

En la figura 17 podemos apreciar el diagrama de flujo correspondiente al proceso “Administración de plugins” que empieza desde el inicio del sistema hasta las operaciones que puede realizar el PluginManager con el registro de los plugins. Al iniciar el sistema el PluginManager crea un registro vacío que posteriormente será llenado con los plugins. Antes de cargar los plugins el Locator busca los archivos con extensión .plugin localizados en el directorio especificado en archivo de configuración del sistema. Con ayuda de este archivo el Loader carga los plugins a un objeto del sistema, del cual se obtiene el nombre como llave principal para almacenarlo en el registro de PluginManager.

Una vez que se hayan registrado todos los plugins, el PluginManager mantiene en espera a la especificación de la operación a realizar. Estas operaciones pueden ser las siguientes:

- Ejecutar el plugin, esta operación corresponde al proceso “Almacenamiento de datos” (Figura 16).
- Enlistamiento de los plugins, esta operación obtiene los nombres de todos los plugins en el registro y genera una lista con cada nombre.
- Mostrar información del plugin, esta operación genera un mensaje con la información general del plugin, la información general se compone del nombre, una descripción, versión y ayuda para su funcionamiento.
- Búsqueda de plugin, esta operación realiza una búsqueda en el registro de plugins con un nombre recibido, si no encuentra el plugin se regresa un objeto vacío en caso contrario regresa el plugin encontrado.

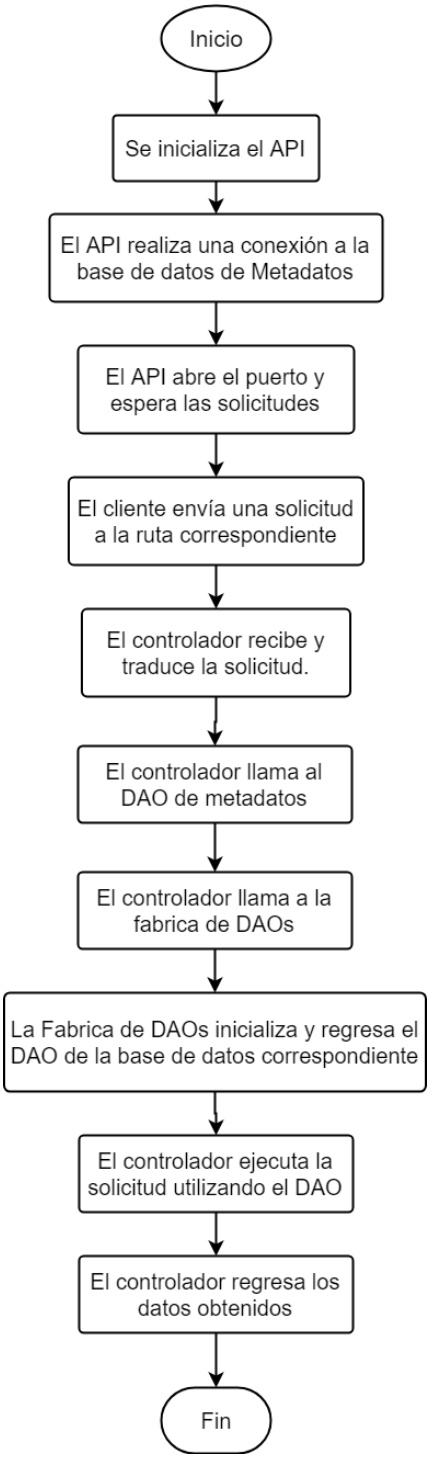


Figura 18. Diagrama de flujo del proceso “Obtención de datos”

En la figura 18 se muestra el proceso “Obtención de datos”, donde al iniciar el API se realiza una conexión con la base de datos de metadatos, estos metadatos corresponden a cada fuente de datos integrada y contienen datos generales de las fuentes como: nombre, esquema, versión y SGBD utilizada. Después de realizar la

conexión satisfactoriamente, el API abre un puerto y espera a la recepción de solicitudes. Las solicitudes serán enviadas por los clientes que utilicen el API a una ruta previamente definida donde el controlador se encargará de recibir y traducir la solicitud.

Una vez que el controlador lea la solicitud, abstrae la identificación de la fuente de datos dentro de la solicitud y, con ayuda del objeto de acceso a datos (DAO), realizará la búsqueda en la base de datos de metadatos. Al obtener los metadatos de la fuente se utilizará el nombre del SGBD para enviarla a la Fabrica de DAOs y así obtener una instancia del DAO que se conectará al SGDB donde los datos integrados se encuentran almacenados. Por último, se le enviará al DAO la solicitud para obtener los datos solicitados por el cliente y serán regresados al cliente.

Vista física

El software se ejecuta en una red de computadoras o nodos de procesamiento. La vista física muestra el mapeo de las redes, procesos, tareas y objetos en los varios nodos. El mapeo del software a los nodos del sistema debe ser flexible y tener un impacto mínimo en el código fuente. En la figura 19 se muestra un diagrama de la vista física de la arquitectura, el o los clientes pueden ser herramientas de visualización y análisis de datos que se encuentran en computadoras separadas a la del servidor web.

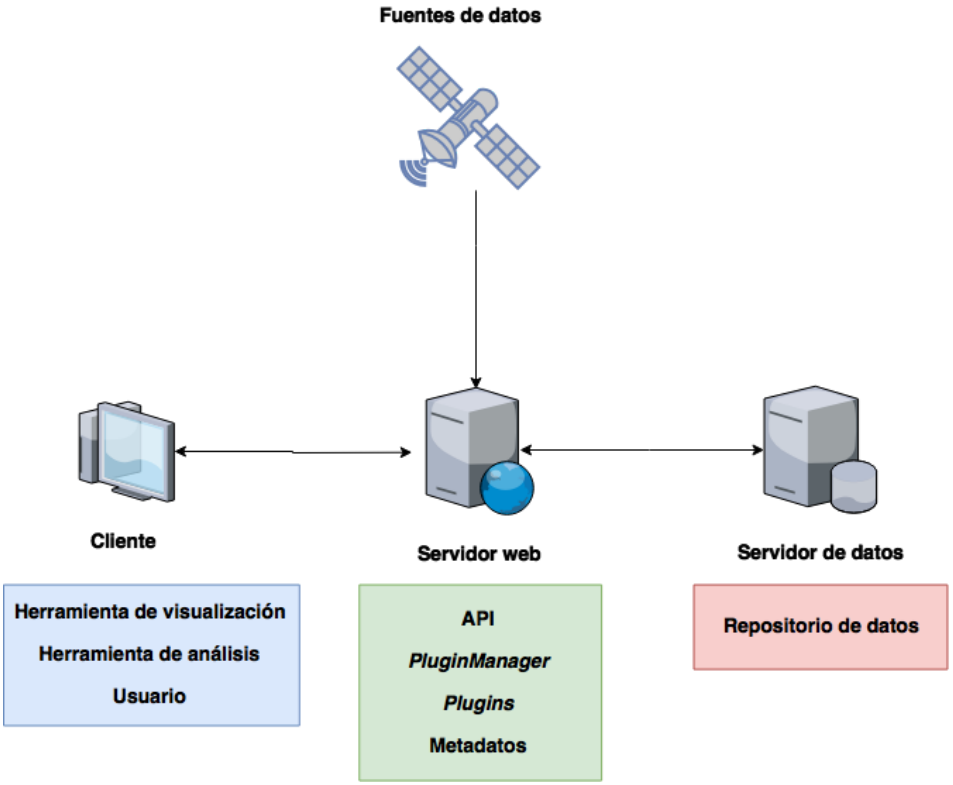


Figura 19. Vista física de la arquitectura propuesta

El servidor web es donde se encuentra hospedada la arquitectura, algunos de los elementos en este servidor son: el API, PluginManager, plugins y la base de datos de los metadatos. Las fuentes de datos cargan los datos a través de los plugins, las fuentes de datos son externas al servidor web y solo envían los datos. El servidor de datos es donde se encuentran los distintos SGDBs con los datos ya integrados.

Procesos

Una vez que se comprende la arquitectura propuesta se definieron los procesos que está soporta desde la perspectiva de los roles que deben interactuar para lograr integrar una nueva fuente de datos y obtener los datos integrados. Para integrar nuevos conjuntos de datos a la arquitectura primero se debe realizar el proceso de importación. La importación se logra con el desarrollo y ejecución de un *plugin* que

se ajuste a la medida de las características del nuevo conjunto(s) de datos a integrar.

El *plugin* desarrollado obtendrá los datos de su fuente en el formato y estructura original y los reestructurará y almacenará en la arquitectura, de tal manera que permita su recuperación eficiente para las consultas más comunes. De esta forma no sólo se elimina la necesidad de aprender a utilizar las herramientas para trabajar con el formato original de los datos, también éstos se almacenan en una estructura que permite la recuperación de subconjuntos de esos datos de forma eficiente.
Este enfoque de *plugin* tiene la ventaja adicional de que, una vez que el plugin se conecte a la arquitectura, éste puede ser utilizado cuantas veces sea necesario para importar datos de las fuentes que éste soporte. Puede ser que un *plugin* sea desarrollado específicamente para ser utilizado en la importación de un conjunto de datos una única vez, o puede ser utilizado para importar de manera incremental datos generados por la fuente en intervalos definidos.

Una vez que los datos se encuentren integrados es posible obtenerlos por medio de las APIs desarrolladas para ello. Estas APIs deben ser utilizadas por las aplicaciones de software que implementen alguna funcionalidad que conlleve al consumo de un subconjunto de los datos almacenados. Al proceso de extraer datos almacenados en la arquitectura se le identifica como exportación, y éste comprende desde el desarrollo de la aplicación que utilice las APIs de acceso para extraer datos (cliente), hasta su despliegue y ejecución.

Las APIs de acceso deben ser lo suficientemente robustas y flexibles para permitir seleccionar un subconjunto de datos, ya sea por área, período, o cualquier otra variable. Debe también ser posible cambiar la estructura de los datos hasta cierto punto y poder también elegir entre diferentes formatos de salida para los datos (por ejemplo CSV, ascii).

El tener más de una opción de exportación a disposición de cualquier persona interesada en los datos simplifica la obtención de estos, ya que no es necesario tener conocimiento sobre el formato original o las herramientas necesarias para manipularlo, enfocándose así al procesamiento que se realizará con esos datos, el subconjunto específico requerido, y el formato que mejor se adapte a las herramientas que el interesado en los datos ya domina, eliminando el problema de la heterogeneidad de los datos y herramientas de manipulación. Antes de definir los procesos de importación e importación se deben definir los roles que interactúan en estos procesos para facilitar la descripción de los procesos.

Roles

Se identificaron nueve roles que participan en los procesos anteriormente mencionados, los cuales van desde Experto en la fuente de datos hasta el Administrador de la arquitectura. A continuación, se describen cada uno de ellos:

Experto en la fuente de datos: Puede ser un técnico de una estación, el responsable de un servidor de archivos o simplemente quien conozca dónde obtener datos o conjuntos de datos de alguna fuente en especial, implicando que

tiene cierto nivel de conocimiento en un área específica. Conoce el proceso de generación de los datos.

Experto en los datos: Es el experto en entender los datos presentados en un conjunto, pudiendo generar información y conocimiento útil a partir de éstos. Se le podría llamar de otro modo, el experto de un área, ya que podría conocer los valores y su significado, además de algunos términos y definiciones que giran alrededor de los datos de un área específica. Es experto en interpretar los datos y en el procesamiento que se requiere aplicar a ellos para que sean utilizables, además de contar con la habilidad de desarrollar algoritmos para la transformación de los datos.

Analista: Experto en dialogar con los clientes y especificar requerimientos de software a partir de necesidades expuestas por los clientes. Adicionalmente deberá tener habilidad para el análisis de los datos y sus fuentes. Analista de software y datos.

Desarrollador: Tiene habilidades de programación y análisis de datos, debe manejar varios lenguajes de script, también debe manejar el lenguaje de programación *Java*. Debe también tener conocimiento de expresiones regulares y procesamiento de archivos de texto y de otros formatos de datos. Debe poder construir código eficiente en el manejo de los recursos computacionales. Conocimiento necesario para acceder a las bases de datos de la arquitectura.

Diseñador: Tiene habilidades para crear diagramas de clases, modelos de diseño de software, diagramas de secuencias, diagramas de estado, etc. y todos aquellos diagramas que se requieran para describir la arquitectura y comportamiento de un *plugin* o un cliente. Estos diagramas están a nivel del lenguaje de implementación.

Diseñador de base de datos: Experto en modelado de datos y en construcción de bases de datos tanto relacionales como no relacionales.

Probador: Realiza las pruebas de carga de datos directamente a la base de datos y también las pruebas de integración del *plugin* y la prueba de carga de datos desde el *plugin*. También realiza las pruebas necesarias al cliente de software.

Encargado de proyecto: Es el encargado del proyecto de desarrollo de software, ya sea la construcción de *plugin* o la construcción de un cliente de software nuevo.

Administrador de la arquitectura: Es el encargado de dar mantenimiento y administrar la arquitectura. Registra nuevos *plugins*, los elimina y los ejecuta.

Proceso de Importación

Este proceso (Figura 20) consiste, descrito de forma general, en la serie de actividades realizadas por los distintos roles involucrados para almacenar un nuevo conjunto de datos en la arquitectura. Como ya se mencionó en la Descripción de la Arquitectura, esta se compone de un middleware basado en *plugins*. Cada *plugin* es desarrollado para ejecutar el proceso de importación de un conjunto de datos en especial.

Cuando se requiere almacenar datos de un conjunto no existente en las bases de datos de la arquitectura, se requiere realizar las actividades de este proceso para que los datos sean almacenados. Básicamente este proceso consiste en el desarrollo y ejecución de un *plugin* que importe los datos de interés a las bases de datos de la arquitectura.

El proceso de importación abarca los siguientes subprocesos:

1. Levantar requerimientos de datos
2. Obtener conjuntos de datos
3. Analizar conjuntos de datos
4. Diseñar *plugin*
5. Diseñar esquema
6. Implementar esquema
7. Implementar *plugin*
8. Realizar pruebas de carga directa
9. Realizar pruebas de integración
10. Realizar pruebas de carga por *plugin*
11. Validar *plugin*
12. Registrar *plugin*
13. Ejecutar *plugin*

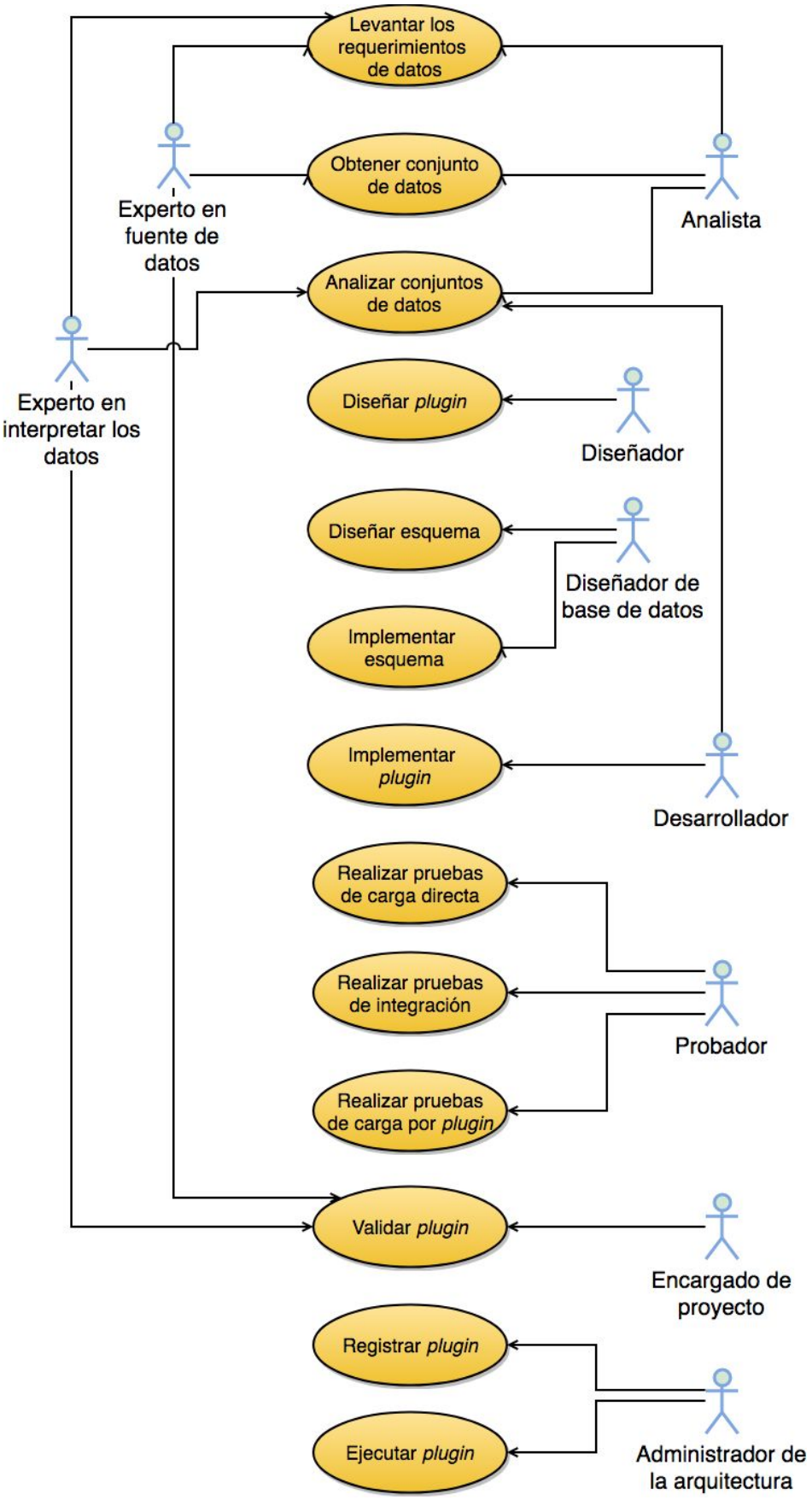


Figura 20. Relación de roles con los subprocesos del proceso de importación

Estos 14 subprocesos fueron identificados, descritos y agrupados según la metodología para almacenar datos en un data warehouse, llamada ETL (Extract, Transform and Load). En la extracción se agruparon los subprocesos 1 y 2. En la transformación se agruparon los subprocesos 3 - 7 y en la carga los subprocesos 8 - 13.

Todos los subprocesos mencionados tienen como finalidad construir un *plugin* que se pueda agregar a la arquitectura para importar uno o más conjuntos de datos nuevos. Un *plugin* es una unidad funcional de software que cumple con una estructura básica designada para que pueda integrarse a la arquitectura. En este caso la función de los *plugins* a integrar es la de importación de datos.

1. **Levantar requerimientos de datos:** Proceso en el cual el analista, el experto en la fuente de datos y el experto en los datos llegan a un acuerdo para los requerimientos que tendrá que cumplir los datos a almacenar dentro de la arquitectura. Ya sea la información o preguntas que se desean contestar o simplemente los datos que se desean almacenar, para lo cual ambos expertos deben de aportar sus conocimientos.
2. **Obtener conjuntos de datos:** Obtener los conjuntos de datos ya sea de forma manual o automática. Ejemplo, descargar de un servidor ftp, de una agencia, sensor, o descargarlo a manera de stream desde una fuente de datos. En este caso interviene el experto en la fuente de datos (el que sabe cómo y dónde se generan), el desarrollador y el analista. Si se desea obtener los datos de forma automática entonces el desarrollador deberá implementar una forma de realizar dicha obtención.

3. **Analizar conjuntos de datos:** El analista analizará la estructura y formato de los conjuntos de datos obtenidos para entender cómo están compuestos y cómo es posible almacenarlos. El Experto en los datos intervendrá ocasionalmente cuando su experiencia respecto a los datos sea requerida por el analista.
4. **Diseñar *plugin***: Diseñar la estructura del plugin a implementar para cargar los conjuntos de datos. Consiste en generar diagramas de clases, secuencias, paquetes, y todos aquellos diagramas necesarios para que el desarrollador pueda implementar la funcionalidad deseada. El diagrama de modelo de datos no se realiza en este proceso.
5. **Diseñar esquema:** Este proceso es exclusivo al diseño y modelado del esquema que los datos a almacenar necesitan y que se debe implementar. El modelo de datos es diagramado por el diseñador de base de datos con base a los requerimientos especificados por el analista.
6. **Implementar esquema:** Implementa el esquema de base de datos diseñado, ya sea de una base de datos relacional o una base de datos no relacional. El deber del diseñador de base de datos es crear el nuevo repositorio para los datos que se desean almacenar. Pondría a disposición del desarrollador la documentación y las APIs necesarias para poder utilizar dicho esquema al implementar el plugin.
7. **Implementar *plugin*:** Proceso en el cual se construye el código que refleje lo diseñado por el diseñador del *plugin*. El desarrollador debe programar todas las funciones que satisfagan los requerimientos de datos especificados por el analista. A su vez, si se trata de datos de stream también se debe de tomar

en cuenta los mecanismos de obtención y limpieza automática de los datos en caso de que no se haya programado todavía. Documentar todo lo programado.

8. **Realizar pruebas de carga directa:** Realizar varias pruebas de carga con los datos a almacenar en una base de datos de prueba, con estructura exacta a la que se va a crear en la arquitectura, esto se realiza con el objetivo de validar que el *plugin* está desarrollado correctamente. La base de datos de prueba se creará en un entorno diferente al de la arquitectura. La prueba de carga es directamente con el script desarrollado.
9. **Realizar pruebas de integración:** Realizar las pruebas de integración del nuevo plugin, registrándose satisfactoriamente en un *plugin* manager de prueba diferente al entorno de ejecución de la arquitectura principal.
10. **Realizar prueba de carga por *plugin*:** Prueba la carga de todos los datos a almacenar a través del *plugin* registrado en el *plugin* manager de prueba. Se registran todos los acontecimientos.
11. **Validar *plugin*:** El experto en los datos observa la información almacenada en la arquitectura de prueba por el *plugin* y valida que ésta sea la que necesite. El experto en la fuente de datos valida los aspectos técnicos de los datos como la precisión de las decimales, etc. El encargado de proyecto entrega la documentación del *plugin* al administrador de la arquitectura y a los expertos.
12. **Registrar *plugin*:** El administrador de la arquitectura registra el nuevo *plugin* probado y validado en la arquitectura principal para proceder a cargar los datos nuevos.

13. **Ejecutar *plugin*:** Ejecuta un nuevo *plugin* registrado para cargar los datos a la arquitectura principal. El esquema debe estar creado en ella previamente, a menos que el *plugin* se encargue de crear el esquema también.

Proceso de exportación

Este proceso (Figura 21) consiste en la serie de actividades a realizar cada vez que se desee implementar una nueva funcionalidad de exportación de datos en la arquitectura. Dichas funcionalidades son implementadas a manera de clientes de software que ofrezcan la exportación, obtención y visualización de los datos, entre otras funcionalidades más. Los clientes de software desarrollados pueden ser tanto aplicaciones de escritorio, móviles, web o cualquier otro tipo de aplicación que utilice las APIs de acceso de la plataforma.

Este proceso consiste básicamente en el desarrollo de software de un nuevo cliente que utilice los APIs de acceso de la plataforma para obtener datos almacenados en ella y realizar cualquier tipo de transformación o procesamiento con ellos. Ejemplos de clientes de software pueden ser: Un cliente web que utilice los datos almacenados de una estación LI-COR en especial para visualizar las variables almacenadas y poder exportar un subconjunto de esas variables a un formato como CSV o, incluso, visualizar gráficas del comportamiento de una variable dada a través del tiempo. Otro ejemplo de cliente de software puede ser una aplicación de escritorio que genere índices climáticos a partir de una serie de variables designadas con anterioridad.

El proceso de exportación abarca los siguientes subprocesos:

1. Levantar requerimientos del cliente
2. Analizar procesamiento de datos
3. Diseñar cliente
4. Implementar cliente
5. Probar cliente
6. Validar cliente
7. Desplegar cliente

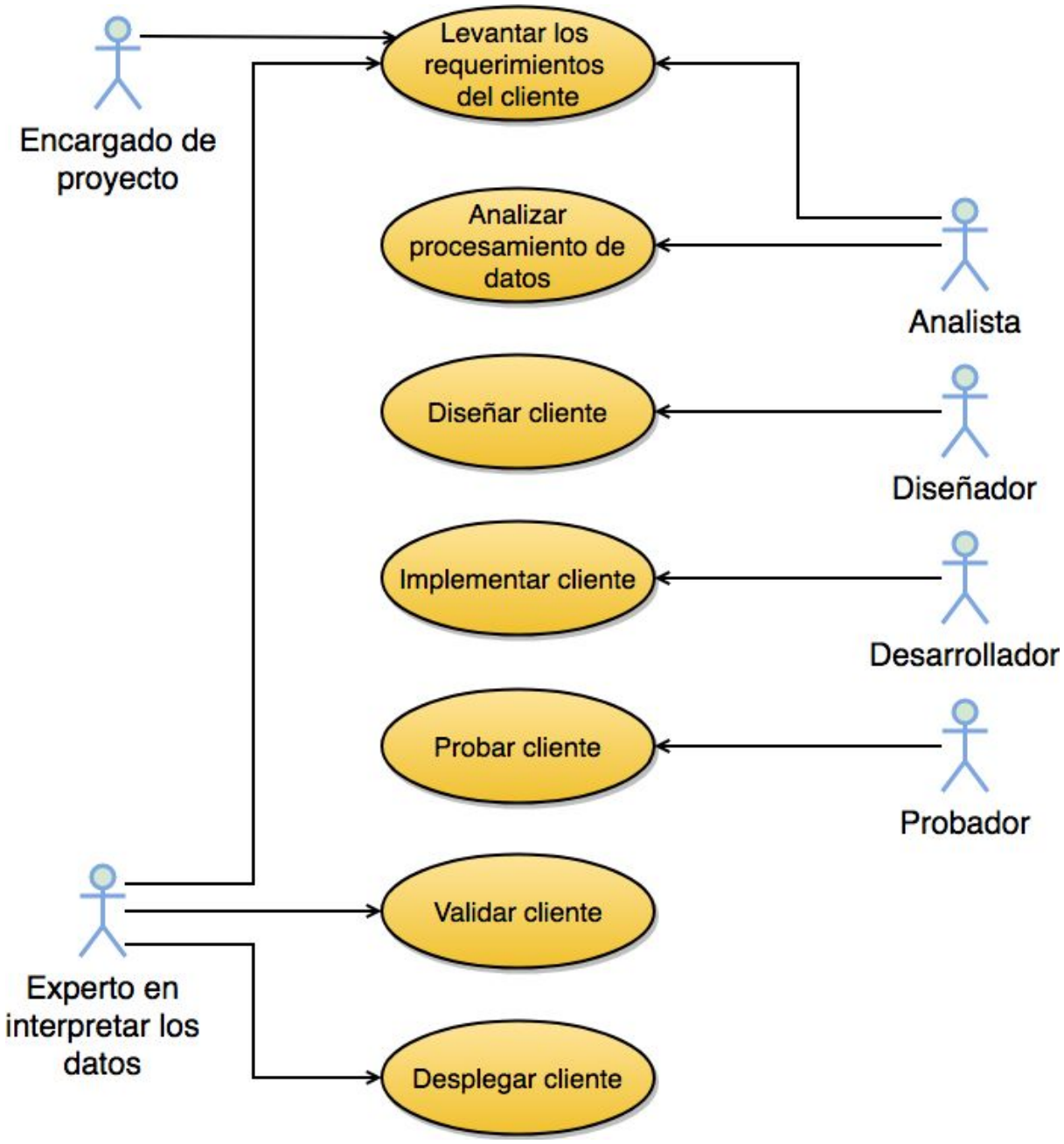


Figura 21. Relación de roles con los subprocesos del proceso de exportación

Estos subprocesos tienen el objetivo de construir un cliente de software que pueda utilizar los APIs de la arquitectura para acceder a los datos almacenados en ella. A continuación se describen cada uno de los subprocesos:

1. **Levantar requerimientos del cliente:** El encargado de proyecto en conjunto con el analista levantarán los requerimientos que el experto en los datos necesita que sean satisfechos por el cliente.
2. **Analizar procesamiento de datos:** Analizar si es necesario realizar algún procesamiento a los datos almacenados antes de presentarlos o ser exportados por el cliente. Para este paso se asume que ya existen los datos necesarios almacenados en la arquitectura, de lo contrario es necesario comenzar con el proceso de Importación.
3. **Diseñar cliente:** Crea todos los modelos y diagramas necesarios en el lenguaje de implementación que cumplan con los requerimientos y el resultado del análisis del procesamiento a los datos que se generaron anteriormente. Esta es la entrada para que el desarrollador implemente lo generado por el diseño.
4. **Implementar cliente:** Programar el cliente diseñado. Utilizar todas las librerías y APIs necesarias, provistas por la arquitectura para recuperar los datos.
5. **Probar cliente:** Realizar las pruebas de funcionalidad, integridad, etc. para verificar que el cliente funcione correctamente y cumple efectivamente con los requerimientos establecidos.
6. **Validar cliente:** Prueba de aceptación por parte del experto en los datos.
7. **Desplegar cliente:** Instalación del cliente ya sea en un servidor o bien un ordenador especificado por el experto en los datos.

Metodología para el desarrollo de *plugin*

Los *plugins* son aplicaciones ligeras con una función en específico, el uso de estos permiten que la arquitectura propuesta sea extensible, ya que se le agregan nuevos *plugins* que se encargan de integrar nuevas fuentes de datos. Para poder integrar un nuevo *plugin* satisfactoriamente en la arquitectura este debe de adoptar los lineamiento establecidos de tal manera que todos los *plugins* se encuentren estandarizados en su ejecución.

Uno de los primeros lineamientos que se deben verificar antes de realizar el desarrollo del *plugin* es el aseguramiento de calidad de los datos. El Experto en los datos debe asegurarse que el proceso de aseguramiento de calidad de los datos haya sido realizado previo al desarrollo de *plugin*, ya que este proceso está fuera del alcance del *plugin*. Esto se debe a que la limpieza de datos es un proceso que puede llegar a convertir al *plugin* en una aplicación compleja mientras que uno de los propósitos del *plugin* es el de mantener una aplicación ligera.

El principal lineamiento que debe cumplir el *plugin* es la implementación de las interfaces establecidos en la arquitectura, estos son: *Plugin, PluginLoader, PluginDescriptor* y *CopyStrategy* y se encuentran definidos en la sección “Vista Lógica” de este capítulo. Con las interfaces implementadas la arquitectura puede comprender cómo interactuar con el *plugin* ya que conoce los métodos que este puede ejecutar.

Otra directriz que debe cumplir el *plugin* es inclusión de un archivo de descripción que defina los metadatos del *plugin*. Estos metadatos le permiten a la arquitectura registrar al *plugin* ya que en el archivo de descripción se define la clase *loader* que utilizará para cargar el *plugin*, además de almacenar los metadatos en una base de datos que puede ser accedida a través de los APIs. Este archivo de descripción debe definirse en el lenguaje de serialización legible para los humanos YAML. Este documento debe definir los siguientes elementos:

- **Nombre:** Nombre del *plugin*
- **Versión:** Número de versión del *plugin*
- **Descripción:** Descripción del *plugin.* Aquí se puede definir el funcionamiento del *plugin,* información de la fuente de datos, estructura de los datos, etc.
- **Script utilizado:** Nombre del *script* si este *plugin* utiliza uno.
- **Archivo de ayuda:** Nombre de archivo de ayuda o auxiliar que contenga información relevante para el usuario del *plugin*.
- **Clase cargadora:** Clase principal del *plugin* que implemente la interfaz *PluginLoader.*
- **Lenguaje de programación utilizado en script:** Lenguaje de programación utilizado por el *script* si este *plugin* utiliza uno.
- **Sistema operativo objetivo:** Sistema operativo donde se puede ejecutar este *plugin.*
- **Sistema gestor de base de datos objetivo:** Nombre del SGBD donde almacenarán los datos.

- **Archivos soportados:** Lista de nombres de los archivos que este *plugin* soporta.
- **Dependencias:** Librerías o módulos que utilice el *plugin* para su ejecución.
- **Creador:** Nombre del desarrollador del *plugin*.

Unos ejemplos de este archivo de descripción se encuentran en las Figuras 23 y 27, estos archivos de descripción corresponden a los *plugins* utilizados en los casos de estudio Querétaro y GHCN respectivamente que seran descritos a continuación.

Casos de estudio

Para validar técnicamente la arquitectura propuesta se establecieron dos casos de estudio, por cada caso de estudio se planeó integrar una fuente de datos distinta siguiendo las directrices de la arquitectura con el motivo de validar que los procesos de “Administración de *plugins*” y “Almacenamiento de datos” se lleven a cabo satisfactoriamente. Además se desarrolló un cliente con interfaz gráfica para validar el proceso “Obtención de datos”, cabe mencionar que el cliente no requiere de una interfaz gráfica pero se decidió desarrollar el cliente con una debido a los requerimientos del caso de estudio.

Caso “Querétaro”

Este caso de estudio se originó del proyecto “Heterogeneidad y escalamiento de los flujos de superficie de agua y energía, suelo-vegetación-atmósfera en sistema climáticos regionales. Altiplano de México” donde se planeó la colaboración entre la Universidad Autónoma de Querétaro (UAQ) y la Universidad Autónoma de Baja

California. La UAQ cuenta con una estación distribuida por LI-COR Biosciences, localizada en Bernal, Querétaro, en las coordenadas 20.717 y -99.941 de latitud y longitud respectivamente (Figura 8).

La estación (Figura 22) cuenta con el instrumento analizador de gases llamado LI-7500RS de la empresa LI-COR. El instrumento está diseñado para mediciones de dióxido de carbono (CO_2) de alta velocidad y vapor de agua en el aire ambiente, además incluye el sistema *SMARTFlux*. Esta es una aplicación que ejecuta el software *EddyPro*, desarrollado por LI-COR, para el procesamiento de los datos que son puestos, a su vez, a disponibilidad por medio del sitio web *FluxSuite*, descrito por la empresa como: "un servicio web seguro que provee resultados e información en tiempo real del sitio de eddy covarianza".

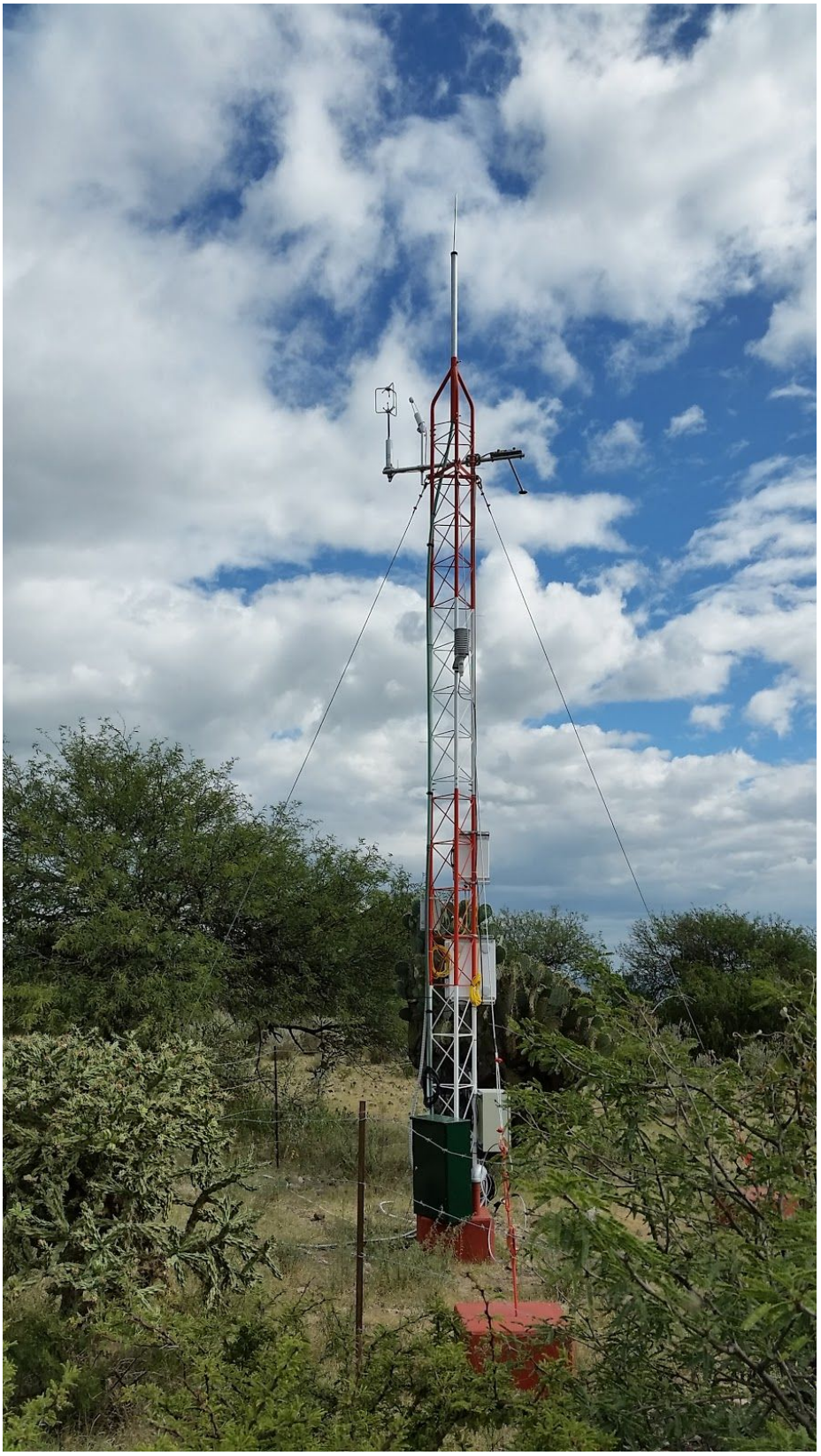


Figura 22. Estación utilizada en Querétaro

Debido a que los datos que son puestos a disponibilidad en el sitio *FluxSuite* son los datos ya procesados por la aplicación *EddyPro*, los datos pre procesados o crudos no se encuentran en este sitio y para acceder a ellos se necesita hacer de manera manual en la estación. Dado que no se cuenta con la posibilidad de obtener los datos pre procesados de manera constante sólo se integrarán los datos procesados localizados en el sitio *FluxSuite*. El sitio *FluxSuite* cuenta con registros desde el día 18 de Agosto del 2016 a la fecha, los datos de este periodo de componen de 204 variables (Tabla 3) y el sitio recibe los datos procesados de la estación cada 30 minutos.

Tabla 3. Variables utilizadas por los datos provenientes de la estación en Querétaro

Etiqueta	**Unidades, Formato o Rango**	**Descripción**
filename	-	Nombre del archivo (o el primero de un conjunto) de datos crudos del cual se extrajo el conjunto de datos para el intervalo de promediación actual
date	yyyy-mm-dd	Fecha del final del período promediado
time	HH:MM	Tiempo del final del período promediado
DOY	ddd.ddd	Día del año

daytime	1=daytime	Bandera para identificar si es de día o no.
file_records	#	Número de registros válidos que se encuentran en el archivo (o conjunto de archivos) de datos crudos
used_records	#	Número de registros válidos utilizados para actualizar el período de promedio
Tau	kg+1m-1s-	Flujo de momento corregido
qc_Tau	#	Bandera de calidad para el flujo de momento
rand_err_Tau	kg+1m-1s-	Error aleatorio para el flujo de momento, si está seleccionado
H	W+1m-2	Flujo de calor sensible corregido
qc_H	#	Bandera de calidad para flujo de calor sensible
rand_err_H	W+1m-2	Error aleatorio para el flujo de calor sensible, si está seleccionado
LE	W+1m-2	Flujo de calor latente corregido

qc_LE	#	Calidad del flujo de calor latente
rand_err_LE	W+1m-2	Error aleatorio para el flujo de calor latente, si está seleccionado
co2_flux	µmol+1s-1m-2	Flujo de co2 corregido
qc_co2_flux	#	Bandera de calidad para *el* flujo de co2
rand_err_co2_flux	µmol+1s-1m-2	Error aleatorio para el flujo de co2, si está seleccionado
h2o_flux	mmol+1s-1m-2	Flujo de h2o corregido
qc_h2o_flux	#	Bandera de calidad para *el* flujo de h2o
rand_err_h2o_flux	mmol+1s-1m-2	Error aleatorio para el flujo de h2o, si está seleccionado
ch4_flux	µmol+1s-1m-2	Flujo de ch4 corregido
qc_ch4_flux	#	Bandera de calidad para *el* flujo de ch4
rand_err_ch4_fllux	µmol+1s-1m-2	Error aleatorio para el flujo de ch4, si está seleccionado
none_flux	µmol+1s-1m-2	Flujo de none corregido

qc_none_flux	#	Bandera de calidad para *el* flujo de none
rand_err_none_flux	µmol+1s-1m-2	Error aleatorio para el flujo de none, si está seleccionado
H_strg	W+1m-2	Estimación del flujo de calor sensible almacenado
LE_strg	W+1m-2	Estimación del flujo de calor latente almacenado
co2_strg	µmol+1s-1m-2	Estimación del flujo de co2 almacenado
h2o_strg	mmol+1s-1m-2	Estimación del flujo de h2o almacenado
ch4_strg	µmol+1s-1m-2	Estimación del flujo de ch4 almacenado
none_strg	µmol+1s-1m-2	Estimación del flujo de none almacenado
co2_v-adv	µmol+1s-1m-2	Estimación del flujo de advección vertical
h2o_v-adv	mmol+1s-1m-2	Estimación del flujo de advección vertical

ch4_v-adv	µmol+1s-1m-2	Estimación del flujo de advección vertical
none_v-adv	µmol+1s-1m-2	Estimación del flujo de advección vertical
co2_molar_density	mmol+1m-3	Densidad molar de co2 medida o estimada
co2_mole_fraction	µmol+1mol_a-1	Fracción molar del co2 medida o estimada
co2_mixing_ratio	µmol+1mol_d-1	Proporción de mezcla del co2 medida o estimada
co2_time_lag	s	Retraso de tiempo usado para sincronizar series de tiempo de co2
co2_def_timelag	1=default	Bandera: si el retraso informado es el valor predeterminado (T) o calculado (F)
h2o_molar_density	mmol+1m-3	Densidad molar de h2o medida o estimada
h2o_mole_fraction	mmol+1mol_a-1	Fracción molar del h2o medida o estimada

h2o_mixing_ratio	mmol+1mol_d-1	Proporción de mezcla del h2o medida o estimada
h2o_time_lag	s	Retraso de tiempo usado para sincronizar series de tiempo de h2o
h2o_def_timelag	1=default	Bandera: si el retraso informado es el valor predeterminado (T) o calculado (F)
ch4_molar_density	mmol+1m-3	Densidad molar de ch4 medida o estimada
ch4_mole_fraction	µmol+1mol_a-1	Fracción molar del ch4 medida o estimada
ch4_mixing_ratio	µmol+1mol_d-1	Proporción de mezcla del ch4 medida o estimada
ch4_time_lag	s	Retraso de tiempo usado para sincronizar series de tiempo de ch4
ch4_def_timelag	1=default	Bandera: si el retraso informado es el valor predeterminado (T) o calculado (F)

none_molar_density	mmol+1m-3	Densidad molar de none medida o estimada
none_mole_fraction	µmol+1mol_a-1	Fracción molar del none medida o estimada
none_mixing_ratio	µmol+1mol_d-1	Proporción de mezcla del none medida o estimada
none_time_lag	s	Retraso de tiempo usado para sincronizar series de tiempo de none
none_def_timelag	1=default	Bandera: si el retraso informado es el valor predeterminado (T) o calculado (F)
sonic_temperature	K	La temperatura media del aire ambiente medida por el anemómetro
air_temperature	K	La temperatura media del aire ambiente, calculada a partir de las lecturas de temperatura del aire de alta frecuencia, o estimada a partir de la temperatura sónica

air_pressure	Pa	La presión media del aire ambiente, calculada a partir de lecturas de presión de aire de alta frecuencia, o estimada en base a la altitud del sitio (presión barométrica)
air_density	kg+1m-3	Densidad del aire ambiente
air_heat_capactiy	J+1kg-1K-1	Calor específico a presión constante del aire ambiente
air_molar_volume	m+3mol-1	Volumen molar de aire ambiente
ET	mm	Flujo de evapotranspiración
water_vapor_density	kg+1m-3	Densidad de masa ambiente del vapor de agua
e	Pa	Presión parcial del vapor de agua ambiente
es	Pa	Presión parcial de vapor de agua ambiente a saturación
specific_humidity	kg+1kg-1	Humedad ambiental específica en una base masiva
RH	%	Humedad relativa ambiente

VPD	Pa	Déficit de presión de vapor de agua ambiente
Tdew	K	Temperatura de rocío
u_unrot	m+1s-1	Componente del viento a lo largo del eje *u* del anemómetro
v_unrot	m+1s-1	Componente del viento a lo largo del eje *v* del anemómetro
w_unrot	m+1s-1	Componente del viento a lo largo del eje *w* del anemómetro
u_rot	m+1s-1	Componente de viento *u* girado (velocidad media del viento)
v_rot	m+1s-1	Componente de viento *v* girado (debe ser cero)
w_rot	m+1s-1	Componente de viento *w* girado (debe ser cero)
wind_speed	m+1s-1	Velocidad media del viento
max_wind_speed	m+1s-1	Máxima velocidad instantánea del viento

wind_dir	deg_from_north	Dirección desde la cual sopla el viento, con respecto al norte geográfico o magnético
yaw	° (degrees)	Primer ángulo de rotación
pitch	° (degrees)	Segundo ángulo de rotación
u*	m s-1	Velocidad de fricción
TKE	m+2s-2	Energía cinética turbulenta
L	m	Longitud Monin-Obukov
(z-d)/L	#	Parámetro de estabilidad Monin-Obukhov
bowen_ratio	#	Proporción del flujo de calor sensible con respecto de flujo de calor latente
T*	K	Escala de temperatura
model	0=KJ/1=KM/2=HS	Modelo para la estimación de la huella
x_offset	m	A lo largo de la distancia del viento que proporciona una contribución <1% a los flujos turbulentos

x_peak	m	A lo largo de la distancia del viento que proporciona la contribución más alta (pico) a los flujos turbulentos
x_10%	m	A lo largo de la distancia del viento proporcionando una contribución del 10% (acumulativa) a los flujos turbulentos
x_30%	m	A lo largo de la distancia del viento aportando una contribución del 30% (acumulativa) a los flujos turbulentos
x_50%	m	A lo largo de la distancia del viento proporcionando una contribución del 50% (acumulativa) a los flujos turbulentos
x_70%	m	A lo largo de la distancia del viento proporcionando una contribución del 70%

		(acumulativa) a los flujos turbulentos
x_90%	m	A lo largo de la distancia del viento proporcionando una contribución del 90% (acumulativa) a los flujos turbulentos
un_Tau	kg+1m-1s-2	Flujo de momento no corregido
Tau_scf	#	Factor de corrección espectral para el flujo de momento
un_H	W+1m-2	Flujo de calor sensible no corregido
H_scf	#	Factor de corrección espectral para el flujo de calor sensible
un_LE	W+1m-2	Flujo de calor latente no corregido
LE_scf	#	Factor de corrección espectral para el flujo de calor latente
un_co2_flux	µmol+1m-2 s-1	Flujo de co2 no corregido
co2_scf	#	Factor de corrección espectral para el flujo de co2

un_h2o_flux	mmol+1m-2 s-1	Flujo de h2o no corregido
h2o_scf	#	Factor de corrección espectral para el flujo de h2o
un_ch4_flux	µmol+1m-2 s-1	Flujo de ch4 no corregido
ch4_scf	#	Factor de corrección espectral para el flujo de ch4
un_none_flux	µmol+1m-2 s-1	Flujo de none no corregido
none_scf	#	Factor de corrección espectral para el flujo de none
spikes_hf	8u/v/w/ts/co2/h2o/ch4/none	Banderas duras para las variables individuales para la prueba del punto
amplitude_resolution_hf	8u/v/w/ts/co2/h2o/ch4/none	Indicadores de variables individuales para la resolución de amplitud
drop_out_hf	8u/v/w/ts/co2/h2o/ch4/none	Indicadores de variables individuales para la prueba de abandono
absolute_limits_hf	8u/v/w/ts/co2/h2o/ch4/none	Banderas duras para variables individuales para límites absolutos

skewness_kurtosis_hf	8u/v/w/ts/co2/h2o/ch4/none	Banderas duras para las variables individuales de skewness y kurtosis
skewness_kurtosis_sf	8u/v/w/ts/co2/h2o/ch4/none	Indicadores blandos para variables individuales para la prueba de asimetría y kurtosis
discontinuities_hf	8u/v/w/ts/co2/h2o/ch4/none	Banderas duras para variables individuales para la prueba de discontinuidades
discontinuities_sf	8u/v/w/ts/co2/h2o/ch4/none	Indicadores blandos para variables individuales para la prueba de discontinuidades
timelag_hf	8u/v/w/ts/co2/h2o/ch4/none	Banderas duras para la concentración de gas para la prueba del retraso de tiempo
timelag_sf	8u/v/w/ts/co2/h2o/ch4/none	Indicadores blandos para la concentración de gas para la prueba de retardo
attack_angle_hf	8aa	Bandera dura para la prueba de ángulo de ataque

non_steady_wind_hf	8U	Bandera dura para la prueba horizontal no constante
u_spikes	#	Número de picos detectados y eliminados para variable u
v_spikes	#	Número de picos detectados y eliminados para variable v
w_spikes	#	Número de picos detectados y eliminados para variable w
ts_spikes	#	Número de picos detectados y eliminados para variable ts
co2_spikes	#	Número de picos detectados y eliminados para variable co2
h2o_spikes	#	Número de picos detectados y eliminados para variable h2o
ch4_spikes	#	Número de picos detectados y eliminados para variable ch4
none_spikes	#	Número de picos detectados y eliminados para variable none

head_detect_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
t_out_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
t_in_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
aux_in_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
delta_p_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
chopper_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
detector_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente

pll_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
sync_LI-7200	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
mean_value_RSSI_LI-7200	#	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
chopper_LI-7500	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
detector_LI-7500	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
pll_LI-7500	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
sync_LI-7500	#_flagged_recs	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente

mean_value_LI-7500	#	Valor medio de AGC para LI-7500A y / o LI-7200, si está presente
not_ready_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
no_signal_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
re_unlocked_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
bad_temp_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
laser_temp_unregulated_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
block_temp_unregulated_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
motor_spinning_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
pump_on_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente

top_heater_on_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
bottom_heater_on_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
calibrating_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
motor_failure_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
bad_aux_tc1_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
bad_aux_tc2_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
bad_aux_tc3_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
box_connected_LI-7700	#_flagged_recs	Valor medio de RSSI para LI-7700, si está presente
u_var	m+2s-2	Varianza de la variable u
v_var	m+2s-2	Varianza de la variable v
w_var	m+2s-2	Varianza de la variable w

ts_var	K+2	Varianza de la variable ts
co2_var	--	Varianza de la variable co2
h2o_var	--	Varianza de la variable h2o
ch4_var	--	Varianza de la variable ch4
none_var	--	Varianza de la variable none
w/ts_cov	m+1s-1K+1	Covarianza entre *w* y variable ts
w/co2_cov	--	Covarianza entre *w* y variable co2
w/h2o_cov	--	Covarianza entre *w* y variable h2o
w/ch4_cov	--	Covarianza entre *w* y variable ch4
w/none_cov	--	Covarianza entre *w* y variable none
vin_sf_mean	--	Valor medio de vin_sf
co2_mean	--	Valor medio de co2
h2o_mean	--	Valor medio de h2o
dew_point_mean	--	Valor medio de dew_poin
co2_signal_strength_7500_mean	--	Valor medio de co2_signal_strength_7500
UNNAMED_1_1_1	(‡)	Variable extra

LOGGERPOWER_1_1_1	(‡)	Variable extra
LOGGERTEMP_1_1_1	C	Variable extra
VIN_1_1_1	V	Variable extra
PPFD_1_1_1	µmol/m^2/s	Variable extra
P_RAIN_1_1_1	m	Variable extra
RG_1_1_1	W/m^2	Variable extra
RH_1_1_1	%	Variable extra
RN_1_1_1	W/m^2	Variable extra
SHF_1_1_1	W/m^2	Variable extra
SHF_2_1_1	W/m^2	Variable extra
SHF_3_1_1	W/m^2	Variable extra
SHFSENS1_1_1_1	OTHER	Variable extra
SHFSENS2_1_1_1	OTHER	Variable extra
SHFSENS3_1_1_1	OTHER	Variable extra
SWC_1_1_1	m^3/m^3	Variable extra
SWC_2_1_1	m^3/m^3	Variable extra

SWC_3_1_1	m^3/m^3	Variable extra
TA_1_1_1	K	Variable extra
TS_1_1_1	K	Variable extra
TS_2_1_1	K	Variable extra
TS_3_1_1	K	Variable extra
VERSION_1_1_1	OTHER	Variable extra
CHK	186	Variable extra

El primer paso en la integración de esta fuente de datos es el desarrollo del *plugin*, el cual fue llamado “Resumen de Querétaro” debido a que los datos que se obtienen son datos procesados y resumidos de los datos crudos. El *plugin* “Resumen de Querétaro” obtiene los datos del sitio web *FluxSite* para almacenarlos empleando como sistema gestor de base de datos *MongoDB*. En la figura 23 se muestra el archivo de descripción utilizado por el *plugin*, aquí se definen los metadatos del *plugin* como el SGBD o las dependencias necesarias.

```
plugin:
  name: QueretaroSummaryPlugin
  version: 1.0.0
  description: >
    Plugin para la descarga de archivos de resumen provenientes de
    la pagina fluxsuite.com, el resumen de los datos proviene de la
    estacion EddyPro localizada en Bernal, Queretaro.
  script:
    - /script/licor_summary.py
  help-file: /script/help/README.txt
  loader-class: impl.load.Loader
  programming-language:
    name: Python
    version: 2.7
  target-OS:
    - Windows
    - Linux
  target-DB: MongoDB
  supported-files:
    - format:
        - txt
    - name: MongoDB
      type: DB
    - name: python
      type: command
    - name: log4j-api-2.5.jar
      type: jar
    - name: log4j-core-2.5.jar
      type: jar
    - name: dataplugin.api_2.0.0.jar
      type: jar
    - name: pymongo
      type: python-module
    - name: ujson
      type: python-module
  vendor: aherrera
```

Figura 23. Archivo de descripción del *plugin* QueretaroSummaryPlugin

Este *plugin* fue desarrollado en lenguaje de programación *Java* 6 para utilizar las interfaces a implementar de la arquitectura, además se desarrolló un *script* o archivo de órdenes. Este *script* fue desarrollado en el lenguaje de programación *Python* versión 2.7 con el motivo de procesar el archivo de datos. El script utiliza la librería

BeautifulSoup 4, esta librería recupera el lenguaje estándar para la creación de páginas web, Lenguaje de Marcas de Hipertexto o HTML y lo almacena en una estructura de árbol para facilitar su lectura en *Python*. Una vez que se obtiene el HTML del sitio *FluxSuite* se obtiene una ficha de autenticación oculta en HTML, esta ficha es una cadena que permite ejecutar solicitudes de descarga de archivos al sitio *FluxSuite*. En las solicitudes hay que proveer la ficha de autenticación y un rango de fechas, este último determina el rango de tiempo de registros incluidos en el archivo a descargar. Los archivos generados son archivos de texto simple (.TXT) con una estructura de valores separados por espacios.

Con la posibilidad de hacer solicitudes al sitio *FluxSuite*, el *plugin* utiliza la librería *requests* para realizar solicitudes web a un servidor. Una de las funcionalidades que brinda esta librería es la capacidad de leer la respuesta de una solicitud como *stream*, el cual es una secuencia de datos disponibles a lo largo del tiempo. La respuesta es enviada en partes, lo que nos permite procesar pequeñas piezas de la respuesta en lugar de esperar que toda la respuesta se recibida. El procesamiento de datos en *stream* nos permite mejorar el procesamiento de los datos debido a que no se requiere mantener toda la respuesta en memoria mientras se procesa.

Cada parte de la respuesta de la solicitud del servidor *FluxSuite* es procesada iterativamente, en este caso se especificó en el código de *Python* que cada parte tuviera un tamaño de 4.096 Kb. Se seleccionó esta cantidad ya que es mayor a lo almacenado en un línea del archivo de datos, lo que a su vez significa que se asegurará cada iteración contenga los datos suficientes para crear un registro.

Mientras se utiliza el *stream* para mantener una conexión con el servidor, cada pedazo del archivo es almacenado en una estructura de datos de *Python* llamada diccionario que básicamente es una estructura de datos clave-valor. Debido a que *MongoDB* trabaja con documentos con estructura de JSON se utilizó una librería llamada *UltraJSON*, un codificador y decodificador de JSON, un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Con la librería se codificó el diccionario generado para posteriormente escribirlo en un archivo JSON. Con el archivo JSON creado el *plugin* utilizó las estrategias de almacenamiento (*CopyStrategy*) de la arquitectura para realizar la carga del archivo a la base de datos *MongoDB*.

El cliente de la base de datos “Resumen de Querétaro” consulta los datos de la base de datos y los descarga. En la Figura 22 se muestra la página principal donde se le da una bienvenida al usuario, en la parte superior se encuentra la barra de navegación que muestra las diferentes opciones de navegación que existen en el cliente: Inicio, Estación, Metadatos y Datos.

Figura 22. Página principal del cliente

En la Figura 23 se muestra la página en la sección “Estación” donde se muestran los detalles de la estación utilizada. Los datos de la estación que se muestran son los siguientes: Nombre, Instituto, localización, latitud/longitud, elevación y los instrumentos con los que cuenta la estación.

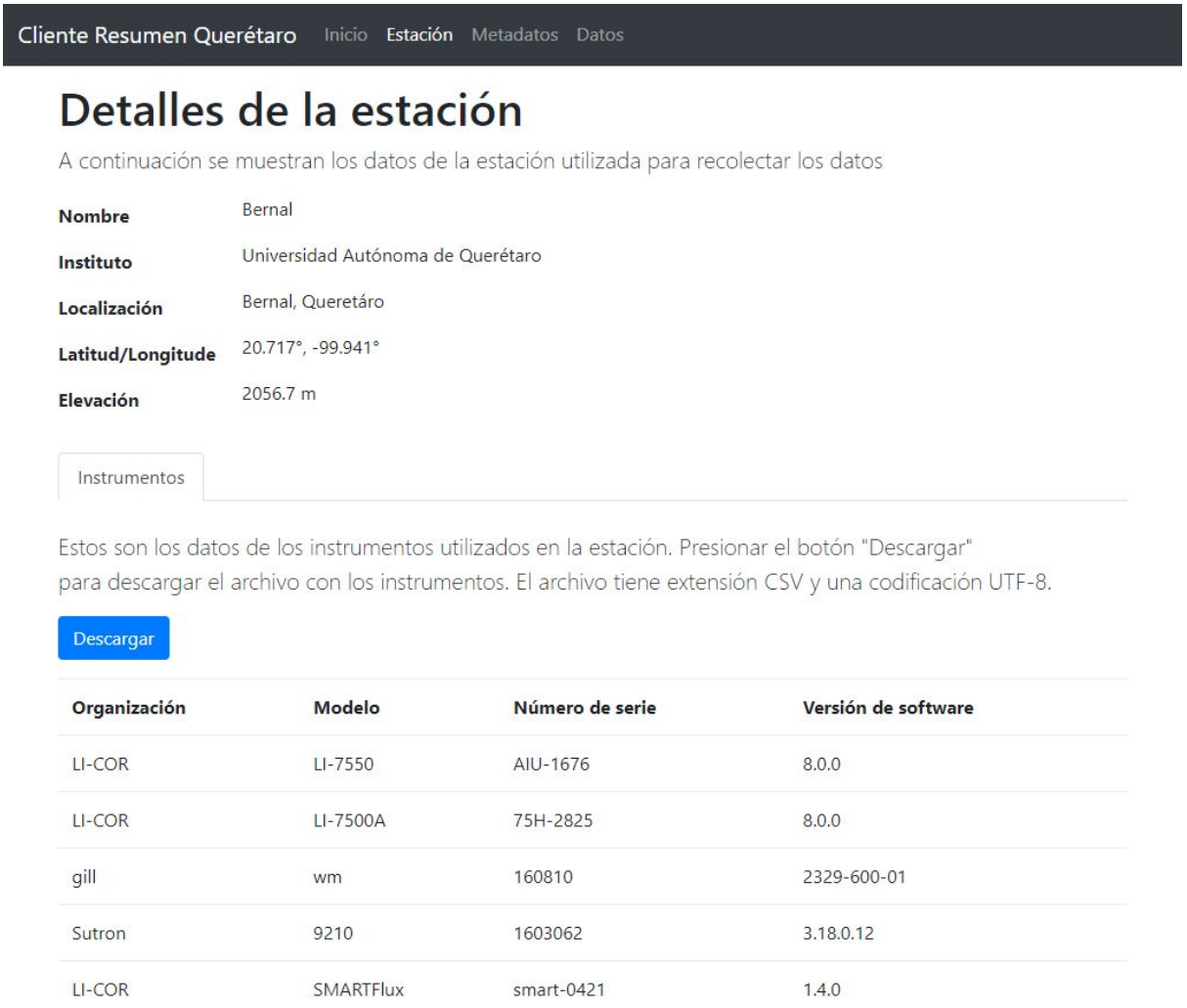


Figura 23. Sección “Estación”

En la Figura 24 se muestra la página en la sección “Metadatos” donde se muestran los detalles de la base de datos utilizada. Los metadatos que se muestran son los siguientes: Nombre, Creado por, fecha de creación, última actualización y los datos del *plugin*.

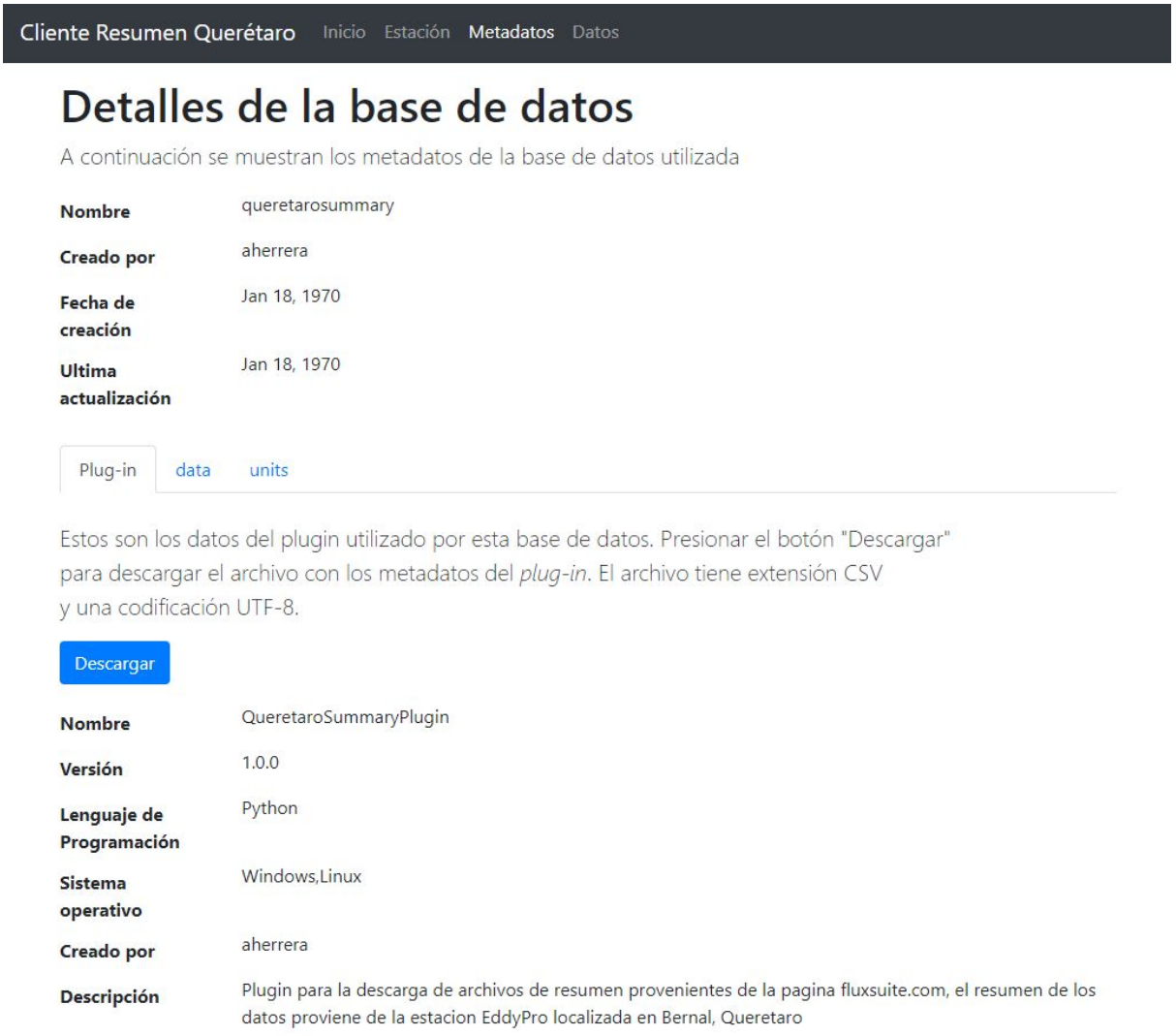
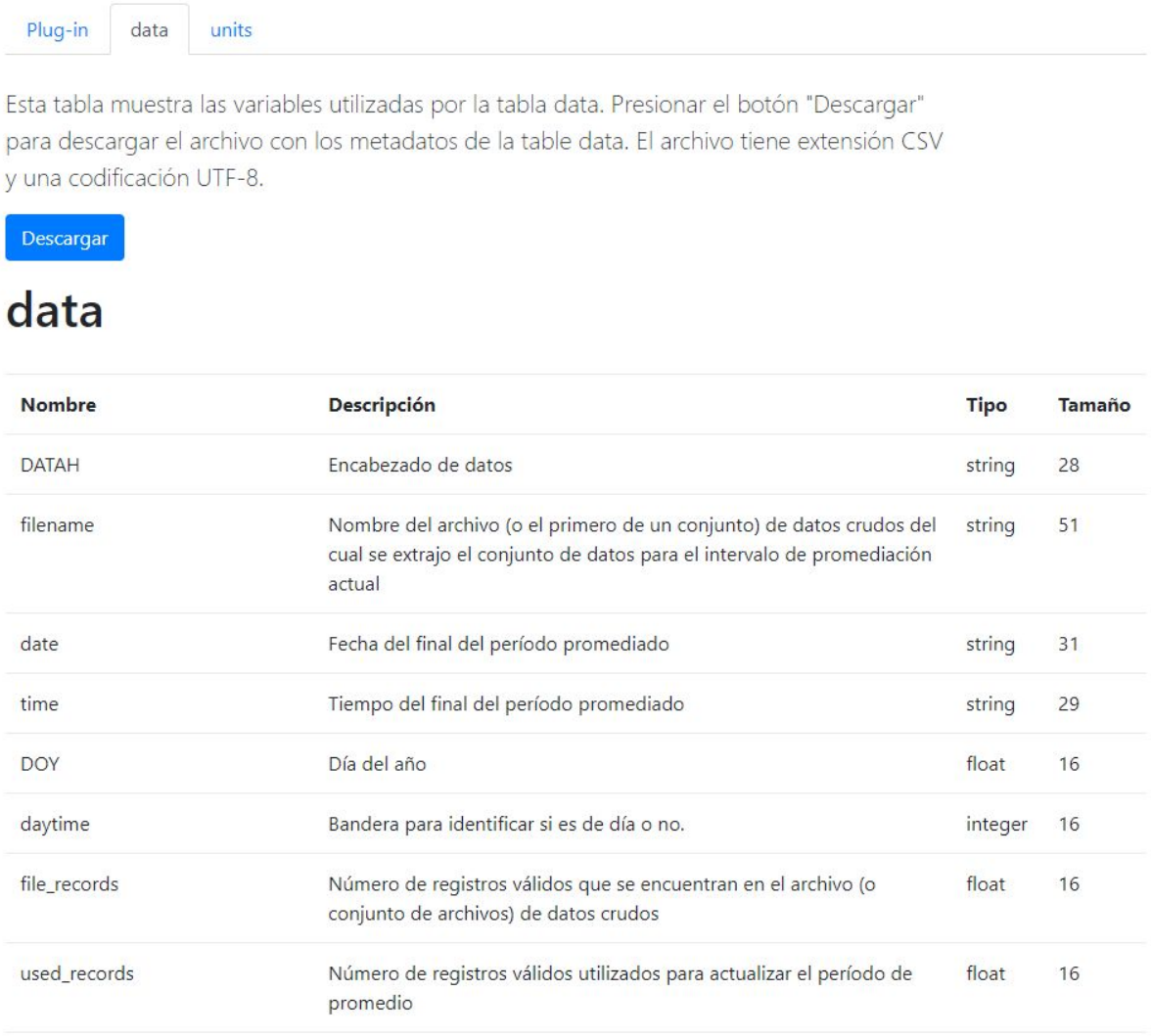


Figura 24. Sección “Metadatos”

Además en las pestañas después de la pestaña de *plugin* se muestra una pestaña por cada tabla o colección de la base de datos. En estas pestañas se se describen el nombre, descripción, tipo y tamaño de cada variable en la tabla (Figura 25). Cabe mencionar que los metadatos de la base de datos son obtenidos por medio de los métodos del API, específicamente el método GET /db/:id/metadata.



Plug-in | data | units

Esta tabla muestra las variables utilizadas por la tabla data. Presionar el botón "Descargar" para descargar el archivo con los metadatos de la table data. El archivo tiene extensión CSV y una codificación UTF-8.

Descargar

data

Nombre	Descripción	Tipo	Tamaño
DATAH	Encabezado de datos	string	28
filename	Nombre del archivo (o el primero de un conjunto) de datos crudos del cual se extrajo el conjunto de datos para el intervalo de promediación actual	string	51
date	Fecha del final del período promediado	string	31
time	Tiempo del final del período promediado	string	29
DOY	Día del año	float	16
daytime	Bandera para identificar si es de día o no.	integer	16
file_records	Número de registros válidos que se encuentran en el archivo (o conjunto de archivos) de datos crudos	float	16
used_records	Número de registros válidos utilizados para actualizar el período de promedio	float	16

Figura 25. Variables de cada tabla.

La sección “Datos” (Figura 26) es donde se presenta la función principal del cliente. En esta sección se encuentra un formulario donde se especifican los elementos necesarios para obtener un subconjunto de los datos almacenados y descargarlos en un formato especificado por el usuario. A continuación se describirán los elementos especificados en el formulario.

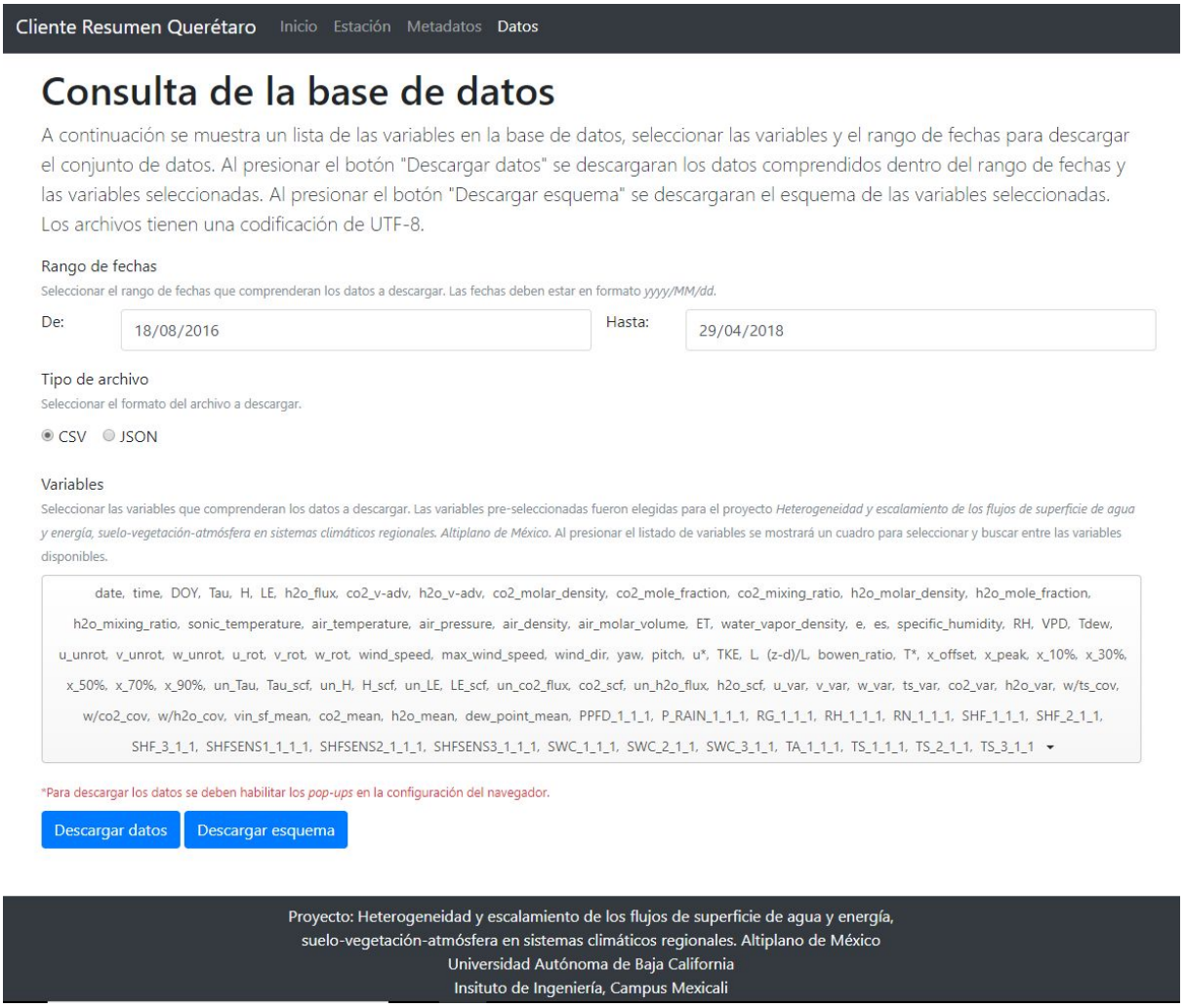


Figura 26. Sección “Datos”.

En la figura 27 se presentan dos elementos del formulario, el rango de fechas nos permite segmentar el conjunto de datos almacenados en un periodo de tiempo. El periodo de tiempo a seleccionar debe encontrarse posterior al día 17 de Agosto del 2016 ya que este es el primer día con el que se cuentan datos. El tipo de archivo especifica en qué formato serán estructurados los datos una vez que se descarguen. Las opciones disponibles son: un archivo separado por comas (.CSV) y un archivo de notación de objetos javascript (JSON).

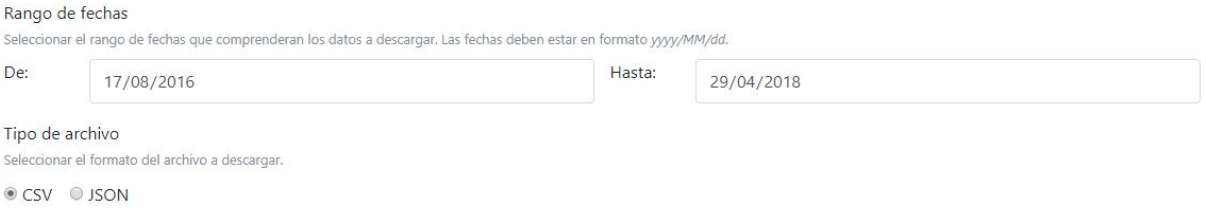


Figura 27. Rango de fechas y tipo de archivo del formulario en la sección “Datos”

El elemento “Variables” (Figura 28) le permitirá al usuario seleccionar las variables que le interese incluir en los datos a descargar. Las variables seleccionadas predeterminadamente fueron establecidas previamente por el expertos en los datos ya que estos son las variables que le interesan.

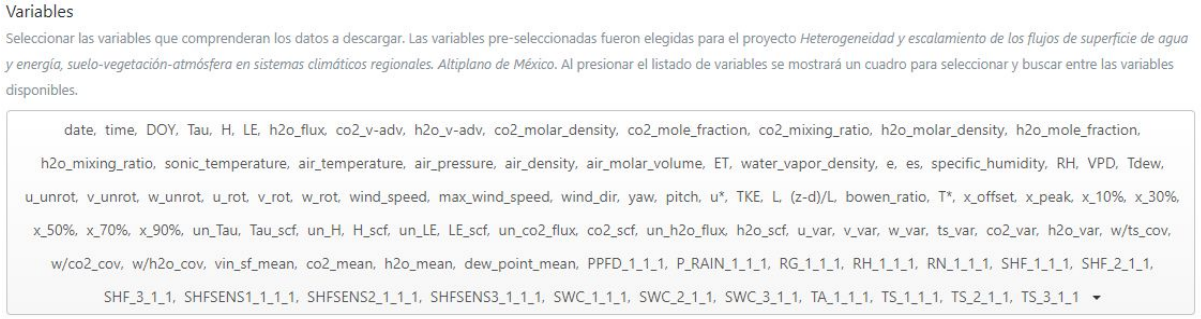


Figura 28. Variables del formulario en la sección “Datos”

El elemento gráfico *angular-multi-select* (Figura 29) le permitirá al usuario seleccionar entre las variables predeterminadas o las no utilizadas para escoger las que le interesen. Debido a la gran cantidad de variables el elemento *angular-multi-select* le permite al usuario realizar una búsqueda dentro del conjunto de variables para encontrar fácilmente las de su interés.

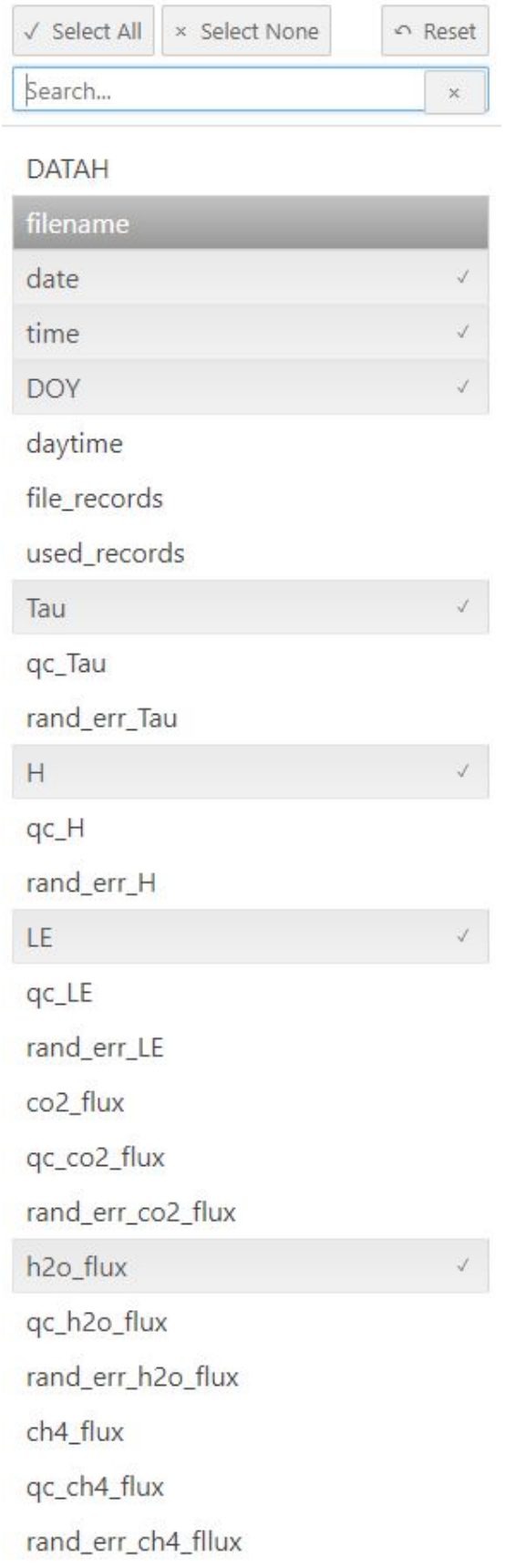


Figura 29. Elemento gráfico *angular-multi-select*

Después de especificar los elementos antes mencionados se debe seleccionar si se desea descargar los datos o el esquema de los datos (Figura 30). Al presionar el botón “Descargar datos” se realizará la solicitud al API para obtener el subconjunto de datos en el periodo, formato y con las variables especificadas. Por otra parte, el botón “Descargar esquema” solicitará un archivo donde se especifica los elementos que componen los datos como: nombre de variables, tipo de dato y tamaño de dato.

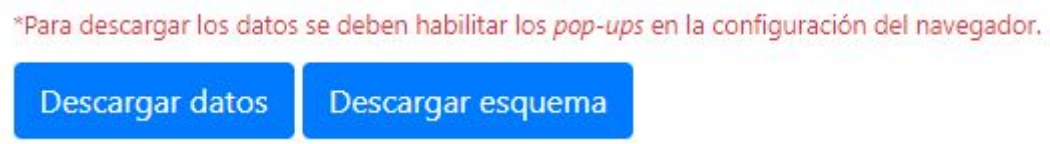


Figura 30. Botones de descarga del formulario en la sección “Datos”

Caso “GHCN”

El objetivo de este caso de estudio es el de integrar, en formato JSON, los datos de las estaciones climatológicas de México, Estados Unidos y Canadá integradas en la base de datos de la “Red Mundial de Clima Histórico” (GHCN por sus siglas en inglés) perteneciente a la Administración Nacional Oceánica y Atmosférica (NOAA por siglas en inglés) de Estados Unidos. El conjunto de las estaciones provenientes de México, Estados Unidos y Canadá constituyen un total de 65,819 estaciones (Figura 31). Esta base de datos es integrada por resúmenes climáticos de estaciones terrestres a lo largo del mundo que han sido sujetas a reseñas de calidad. Algunos de los datos tienen más de 175 años mientras otras menores a una hora. Existen dos tipos de conjuntos GHCN uno con datos diarios y otro con promedios mensuales. El acceso a los datos es utilizado por medio de un servidor FTP o bien por una herramienta para la búsqueda de datos.

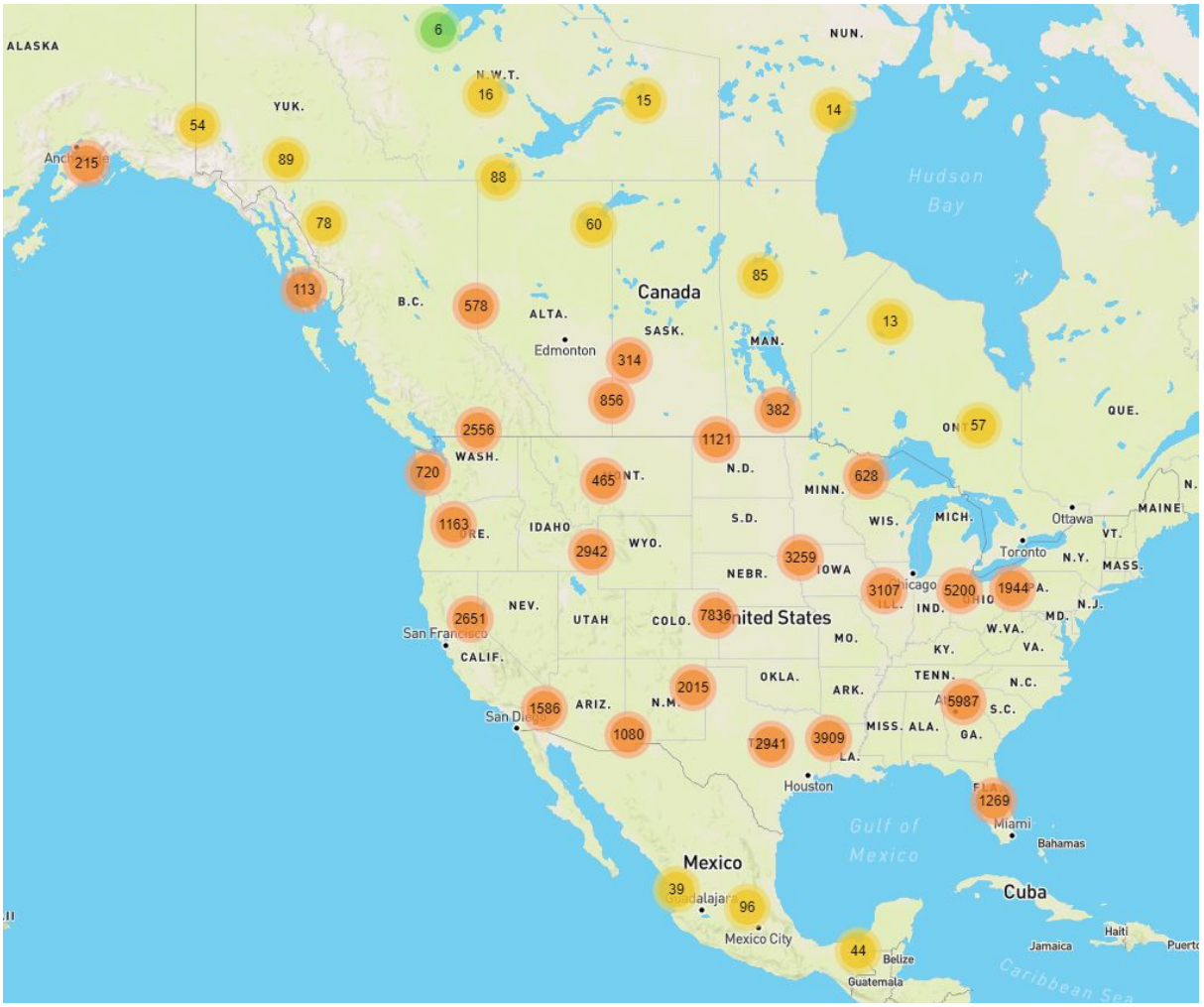


Figura 31. Distribución de las estaciones a o largo de México, USA y Canadá

El *plugin* que se desarrolló para este caso de estudio al igual que el anterior fue desarrollado en el lenguaje de programación *Java* 6 para utilizar las interfaces de la arquitectura y, de la misma manera, se utilizó un *script* desarrollado en *Python* 2.7 para realizar solicitudes al servidor FTP y procesar la respuesta. A continuación en la Figura 27 se definen los metadatos del *plugin* especificados en el archivo de descripción.

```yaml
plugin:
    name: GHCNPlugin
    version: 1.0.0
    description: >
        Plugin para la descarga de los datos pertenecientes
        de las estaciones localizadas en México, USA y Canadá.
        Los datos fueron obtenidos del conjunto de datos "Red
        Climática Histórica Global"(GHCN por sus siglas en ingles).
    script:
        - /script/ghcn_CA_MX_US.py
    help-file: /script/help/README.txt
    loader-class: impl.load.loader
    programming-language:
        name: Python
        version: 2.7
    target-OS:
        - Windows
        - Linux
    target-DB: MongoDB
    supported-files:
        - format:
            - dly
        - name: MongoDB
          type: DB
        - name: python
          type: command
        - name: log4j-api-2.5.jar
          type: jar
        - name: log4j-core-2.5.jar
          type: jar
        - name: dataplugin.api_2.0.0.jar
          type: jar
        - name: ujson
          type: python-module
        - name: requests
          type: python-module
        - name: ftplib
          type: python-module
```

Figura 27. Archivo de descripción del *plugin* GHCN.

El primer paso que se realizó en el *script* es la solicitud del archivo de estaciones (*ghcnd-stations.txt*) que contiene la estructura especificada en la Tabla 4. Con este

archivos se obtienen los metadatos de cada estación que corresponda a los países y son almacenados en una estructura de datos de clave-valor. Además son escritos en un archivo de JSON, llamado *stations.json*, utilizando la librería *UltraJSON* para convertir al estructura clave-valor a JSON.

Tabla 4. Formato del archivo *ghcnd-stations.txt*

Variable	**Índice de columna**	**Tipo de dato**	**Descripción**
ID	1-11	Carácter	Código de identificación de la estación
Latitud	13-20	Número real	Latitud de la estación (grados decimales)
Longitud	22-30	Número real	Longitud de la estación (grados decimales)
Elevación	32-37	Número real	Elevación de la estación (en metros). Valor perdido = -999.9
Estado	39-40	Carácter	Código postal de Estados de Unidos para el estado (solo para estaciones de Estados Unidos)
Nombre	42-71	Carácter	Nombre de la estación
Bandera GSN	73-75	Carácter	Bandera que indica si la estación es parte de la Red de Superficie GCOS.
Bandera HCN/CRN	77-79	Carácter	Bandera que indica si la estación es parte de la Red Histórica Climatológica de Estados Unidos.
ID WMO	81-85	Carácter	El número de estación de la Organización Mundial Meteorológica.

Posteriormente a la obtención de las estaciones se realizan solicitudes a cada archivo con los datos diarios, el nombre del archivo es la identificación de la estación y con la extensión .DLY (Tabla 5). Después de almacenar los datos diarios de la estación estos son escritos en el archivo de JSON, llamado *ghcn.json*. La escritura de los datos en el archivos se realiza por cada estación para evitar la sobrecarga de memoria.

Tabla 5. Formato del archivo *.dly*

Variable	**Índice de columna**	**Tipo de dato**	**Descripción**
ID	1-11	Carácter	Código de identificación de la estación
Año	12-15	Número entero	Año del registro
Mes	16-17	Número entero	Mes del registro
Elemento	18-21	Carácter	Tipo de elemento. Hay cinco elementos principales: - Precipitación (Decimas de milimetros) - Caída de nieve (milímetros) - Profundidad de nieve (milímetros) - Temperatura máxima (décimas de grados celsius) - Temperatura mínima (décimas de grados celsius)
Valor 1	22-26	Número entero	Valor del primer día del mes (perdido = -9999)
Bandera M 1	27-27	Carácter	Bandera de medición del primer día del mes.
Bandera Q 1	28-28	Carácter	Bandera de calidad del primer día del mes.
Bandera S 1	29-29	Carácter	Bandera de fuente del primer día del mes.

Valor 2	30-34	Número entero	Valor del segundo día del mes (perdido = -9999)
Bandera M 2	35-35	Carácter	Bandera de medición del segundo día del mes.
Bandera Q 2	36-36	Carácter	Bandera de calidad del segundo día del mes.
Bandera S 2	37-37	Carácter	Bandera de fuente del segundo día del mes.
			Las últimas cuatro variables se repiten por cada día del mes.

Los archivos creados pueden ser cargados al SGBD utilizando las estrategias de carga (*CopyStrategy*) de la arquitectura. En el caso de este caso de estudio se utilizó el sistema manejador de base de datos MongoDB ya que este manejador cuenta con la posibilidad de crear índices georeferenciados que serán utilizados en el cliente.

El cliente fue desarrollado utilizando el marco de trabajo Angular 4 que contiene los mecanismos y librerías para desarrollar una aplicación web de manera ágil. Los requerimientos del cliente fueron los siguientes: desarrollar una herramienta interactiva para la exploración de estaciones almacenadas y brindar operaciones estadísticas básicas con los datos de cada estación.

En la figura 28 se muestra la vista principal del cliente donde, en el lado derecho, se muestra un mapa con las estaciones almacenadas. La visualización de este mapa se logró con ayuda de la librería Leaflet, una librería de código abierto de *Javascript* para la creación de mapas interactivos. Las estaciones son representadas en el mapa con los marcadores de color azul y con la intención de evitar el ruido al mostrar las estaciones se realiza una agrupación con las estaciones más cercanas

para ser representadas por un círculo con el número de estaciones agrupadas. Una vez se selecciona una estación se muestra en el lado izquierdo los metadato correspondientes a la estación, estos son: identificación, nombre, coordenadas y variables. Además se puede crear una gráfica partir de la variable seleccionada.

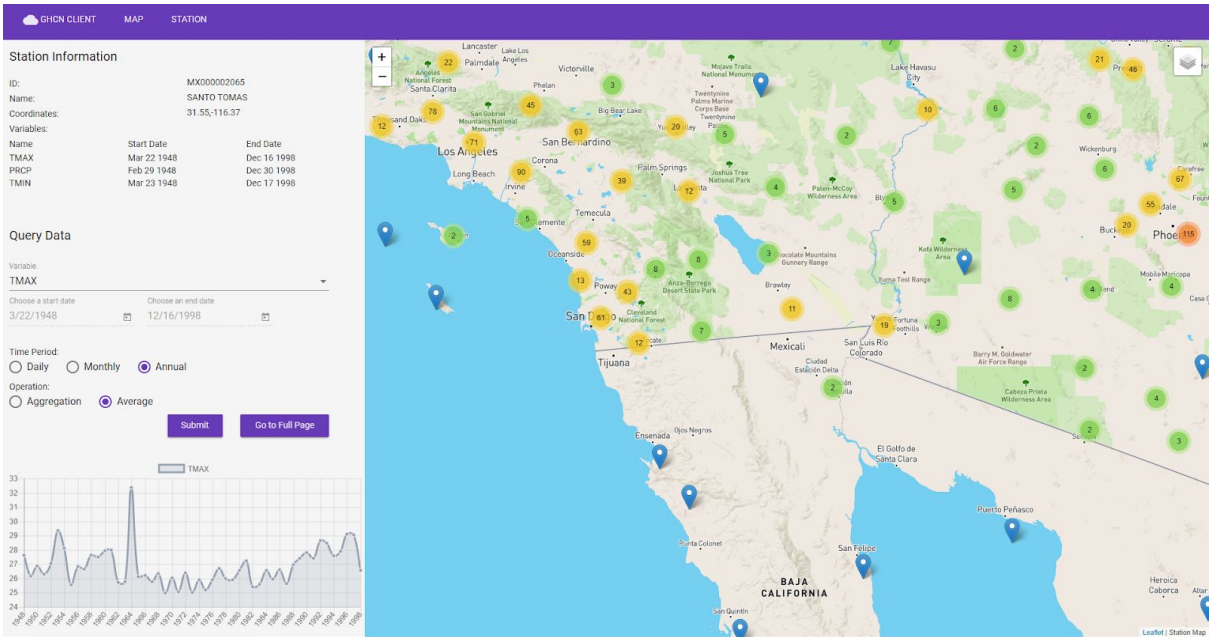


Figura 28. Mapa interactivo de las estaciones

Una de las opciones en el lado izquierdo del mapa interactivo permite al usuario navegar a una sección donde se muestran las operaciones estadísticas básicas, las cuales son: sumatoria, promedio, climatología y anomalías. En la figura 29 se puede apreciar esta sección donde se muestra en la parte superior los metadatos de la estación además de las operaciones ya mencionadas las cuales pueden ser calculadas en base a una unidad de tiempo establecidos: diario, mensual o anual y dentro de un periodo de tiempo dado.

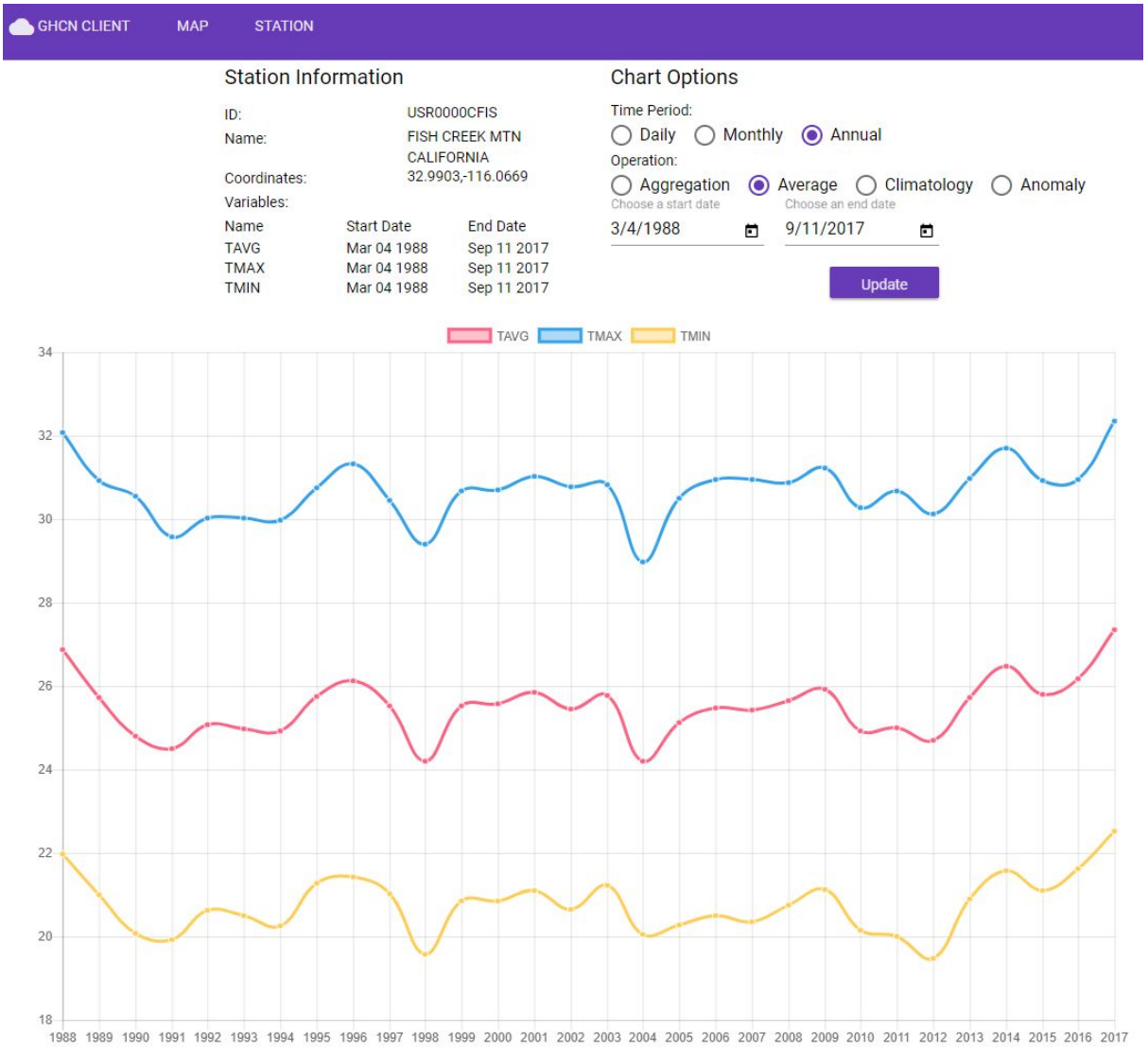


Figura 29. Operaciones básicas en una estación.

Con los datos obtenidos a través de las estaciones se planteó continuar este caso de estudio con la generación de datos en malla derivados de los datos puntuales ya almacenados. Para generar los datos en malla se desarrolló un *script* implementado el algoritmo diseñado por el Dr. Ben Livneh, profesor asistente en la Universidad de Colorado Boulder, con el que se trabajó durante la estancia en la Universidad de Lincoln-Nebraska. El *script* desarrollado utiliza el método de interpolación ponderada

por distancia inversa donde se utilizan por lo menos cuatro estaciones cercanas para calcular el valor en un punto.

Discusiones

La arquitectura propuesta en este proyecto tiene el propósito de resolver los problemas de integración de datos que enfrentan los interesados en conjuntos de datos heterogéneos que no cuentan con los conocimientos computacionales para procesar dichos conjuntos. Mientras que la arquitectura provee la funcionalidad necesaria para integrar nuevos conjuntos de datos además de contar un metodología a seguir para lograrlo, está aún presenta limitaciones que debe enfrentar el interesado.

El interesado en los datos heterogéneos requiere de un experto en computación para desarrollar el *plugin*, esto lo podemos ver en el “Proceso de Importación” donde el rol “Desarrollador” es el que se encarga de programar los *plugins*. La diferencia entre esta arquitectura y las investigadas es el hecho que el interesado cuenta con la metodología de *plugin* que le permitirá seguir un estándar para agregar un nuevo conjunto mientras que en las arquitecturas previas no existían estos lineamientos a seguir.

Otra limitación que debe enfrentar el interesado se encuentra en el aseguramiento de calidad de los datos, ya que este es un proceso que no se encuentra contemplado en al integrar el conjunto de datos con el motivo de mantener el *plugin* ligero y enfocado en una sola funcionalidad. El experto en los datos debe realizar este proceso antes de integrar el conjunto de datos.

Durante el desarrollo de los casos de estudio se presentaron problemas en los procesos realizados. En ambos casos de estudio la obtención de requerimientos se vio afectada por comunicación entre el analista y el experto en los datos, ya que el

experto en los datos en ambos casos de estudio era un experto en el área de meteorología y el analista era un experto en el área computacional.

Una instancia de los problemas de comunicación fue la interpretación de conceptos como “base de datos” donde los expertos en meteorología se refiere a un conjunto de documentos de texto, el analista lo visualizaba como los datos almacenados dentro un SGBD. Para resolver este problema se decidió que al momento de que el experto utilizará un concepto computacional este sea definido para conocer a que se refiere y poder estar en sincronización.

Otro de los problemas en el levantamiento de requerimientos fue la distancia entre el analista y los expertos puesto que uno de los expertos se encuentra en Querétaro y el otro en Nebraska. En el caso de Querétaro se diseñó un cuestionario con las preguntas relevantes para obtener los requerimientos y fue enviado a través de correo electrónico. En el caso del experto en Nebraska se realizó una estancia en Lincoln, Nebraska para poder trabajar de manera cercana en los procesos de importación y exportación.

El conjunto de datos que se utilizó en el caso de Querétaro presentó un problema al momento de obtener los datos debido a que al inicio de lo requerimientos se había establecido que se trabajaría con los datos crudos de la estación, esto probó ser difícil por las circunstancias de la disponibilidad de los datos.

Los datos crudos deben ser obtenidos fisicamente de la estación y posteriormente ser colocados en un servicio de almacenamiento en la nube como DropBox o Google Drive. Sin embargo, la localización de la estación impedía que la obtención de los datos se realizará en un periodo constante. El primer paso que se tomó fue

comunicarse con la distribuidora LI-COR de la estación con intención de conocer algún instrumento que permita enviar los datos crudos.

La distribuidora comentó sobre la instalación de un dispositivo en la estación pero esta se encontraba fuera del presupuesto inicial. Una alternativa fue la de utilizar el sitio web *FluxSuite* donde la estación envía los datos procesados. El conjunto de datos en este sitio web fue el utilizado para integrar a la arquitectura.

Existe dificultad en el aprendizaje de los formatos que utilizan los conjuntos de datos que puede representar un periodo de tiempo extenso donde el analista debe comprender la estructura y funcionamiento de los formatos. En el caso de estudio Querétaro se requirió comprender el formato .GHG donde se investigó que este formato es un archivador como .ZIP o .RAR y consistia de un archivo de datos acompañado de un archivo de metadatos.

Este análisis requiere que el experto en la fuente de datos tenga conocimiento computacionales para explicar al analista como utilizar el formato ya que puede ser un archivador, un archivo de texto, o bien, un archivo binario. Mientras que una de las etapas del análisis de los formatos es el comprender cómo leerlos, el siguiente paso es la comprensión del significado de las variables. Para comprender las variables es necesario del experto en los datos que cuenta con los conocimientos para describir el significado y uso de las variables.

Aunque estas etapas de comprensión de los formatos pueden representar un extenso periodo de tiempo la ventaja se encuentra en el análisis del formato ya que solo se realiza una vez antes de integrar el conjunto posteriormente el interesado o nuevos interesados podrán acceder al conjunto a traves del API sin necesidad de repetir el proceso de comprensión.

Capítulo 5.

Conclusiones

Como resultado de la investigación realizada se encontraron hallazgos relevantes a los campos de estudio, computación y metereología. En cuanto al campo computacional se refiere, el diseño de la arquitectura muestra la complejidad en el tema de heterogeneidad de los datos debido a la integración de cada conjunto de datos requiere de tres expertos, uno en la interpretación de los datos, otro en la fuente de datos y por último, un experto en el campo computacional. Usualmente el interesado en los datos cuenta con la habilidad para interpretarlos pero requiere del experto en la fuente de datos para conocer cómo se generan y su estructura. Si no existiera la arquitectura el experto en computación restructuraría los datos en un formato familiarizado con el interesado. Con el uso de la arquitectura se facilita el análisis de múltiples conjuntos de datos ya que estos se encuentro homogeneizados en un SGBD.

Una de las razones identificadas por la heterogeneidad en el campo de la meteorología es la falta de uso de los estándares establecidos como ISO 12175, ISO 19129, ISO 19130, Binario en malla (GRIB), Forma universal binaria para la representación de datos meteorológicos (BUFR), Estándar de transferencia de datos especiales (SDTS), Formato de datos jerárquicos (HDF), Forma de datos comunes de la red (NetCDF), XML, entre otros. Además de la gran variedad de estándares existen empresas, como LI-COR, que no utilizan los estándares en los instrumentos distribuidos y crean nuevos formatos de datos como .GHG.

En el campo meteorológico existe la necesidad de interoperar los múltiples formatos para que el interesado pueda realizar los procesamientos de interés en los conjuntos de datos ya sea que apliquen o no algún estándar. Si bien la metereología de *plugin* propuesta en este documento no es un estándar en el formato de datos, si se busca estandarizar el proceso de la integración de conjuntos de datos.

Como resultado de los casos de estudio realizados se integraron dos conjuntos de datos, uno correspondiente a los datos en Querétaro y otro al conjunto GHCN. Para lograr interactuar con los datos almacenados se desarrollaron clientes que obtuvieron los datos a través del API. En el desarrollo de los clientes se detectó la necesidad de utilizar herramientas de visualización que permitan no solo mostrar los datos sino que posibiliten la interacción con ellos para abstraer información.

Trabajo a futuro

A continuación se presentan las líneas de investigación que puede ser trabajadas en un futuro. En el caso de este proyecto se decidió dividir las líneas de investigación en dos campos: computación y meteorología, de tal manera que el campo de computación se enfoque en los ámbitos computacionales cómo la arquitectura propuesta y el campo de meteorología se enfoque en el análisis y procesamiento de los conjuntos de datos integrados.

Computación

La arquitectura de plugins propuesta en ese proyecto está basada en la arquitectura propuesta por Dajda (2015) donde propone el concepto de flujo de datos de manera que se cuento con un repositorio de plugins genéricos y estos sean conectados en

un flujo para lograr crear distintos procesamientos en los datos. Por ejemplo se puede tener un plugin enfocado en la obtención de promedios mensuales de datos CSV y otro enfocado en convertir los datos CSV a JSON, al unir ambos plugins se crearía un flujo donde primero se calculan los promedio y posteriormente son convertidos a JSON.

Este concepto no fue implementado en esta iteración de la arquitectura ya que requiere de una investigación más desarrollada donde se conozcan los procesamiento comunes que se realizan en los datos. Con una investigación posterior esta arquitectura puede ser ampliada para incorporar este concepto y lograr un procesamiento más complicados en los datos.

Metereología

Con la capacidad de tener acceso a los datos meteorológicos de manera homogénea se encuentra la posibilidad de realizar análisis comparativos entre distintos conjuntos de datos. Por ejemplo se puede realizar un análisis comparativo en el compartimiento de las estaciones cercanas a la localizada en Querétaro, estas estaciones pueden ser obtenidas del conjunto de datos GHCN. En este caso se realizaría una comparación no solo de estaciones distintas sino de dos conjuntos de datos independientes. Esta clase de comparación puede ser más compleja al integrar nuevos conjuntos de datos.

El conjunto de datos GHCN que fue integrado cuenta con 65,819 estaciones dispersas en México, Canadá y Estados Unidos. Uno de los estudios posibles en

este conjunto es la capacidad de convertir datos puntuales a malla. Debido a la considerable cantidad de estaciones se pueden utilizar para verificar distintos métodos de interpolación y comprobar el índice de error que estos presentan al comparar el resultado con el valor real de las estaciones.

Anexos

Glosario

- Heterogeneidad: El diccionario de la Real Academia Española (RAE) lo define como: “Mezcla de partes de diversa naturaleza en un todo”.

- Heterogeneidad en computación: Wang (2017) define la heterogeneidad en los datos como: “la alta variabilidad en tipos de datos y formatos”

- Heterogeneidad en climatología:

- *Middleware*: Bakken (2001) lo define como: “Una capa de software sobre el sistema operativo pero debajo del programa de aplicación que proporciona una abstracción de programación común a través de un sistema distribuido”

Referencias

1. *2013 IBM Annual Report* [PDF]. (2013). IBM.
2. Zikopoulos, P., & Eaton, C. (2011). *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media.
3. Leser, U. (2000). Query Planning in Mediator Based Information Systems.
4. Beyer, M. A., & Laney, D. (2012). The importance of 'big data': a definition. *Stamford, CT: Gartner*, 2014-2018
5. Gantz, J., & Reinsel, D. (2012, December). THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadow s, and Biggest Growth in the Far East. Retrieved January 23, 2018, from https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf
6. Provost, F., & Fawcett, T. (2013). Data science and its relationship to big data and data-driven decision making. *Big Data*, *1*(1), 51-59.
7. Doan, A., Halevy, A., & Ives, Z. (2012). *Principles of data integration*. Elsevier.
8. Convey, C., Karpenko, O., Tatbul, N., & Yan, J. (2001). Data integration services. *Brown University, New York*.
9. ISO, I. (2011). IEEE: ISO/IEC/IEEE 42010: 2011-Systems and software engineering–Architecture description. *Proceedings of Technical Report*.
10. Inmon, W. H. (2005). *Building the data warehouse*. John wiley & sons.
11. Kimball, Ralph, and Margy Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 2011.
12. Naeem, M. A., Dobbie, G., & Webber, G. (2008, September). An event-based near real-time data integration architecture. In *Enterprise Distributed Object Computing Conference Workshops, 2008 12th* (pp. 401-404). IEEE.
13. Zhu, F., Turner, M., Kotsiopoulos, I., Bennett, K., Russell, M., Budgena, D., ... & Xu, J. (2004, July). Dynamic data integration using web services. In *Web Services, 2004. Proceedings. IEEE International Conference on* (pp. 26)

14. Dajda, J., & Dobrowolski, G. (2015, March). Architecture dedicated to data integration. In *Asian Conference on Intelligent Information and Database Systems* (pp. 179-188). Springer, Cham.
15. Spellman, F. R. (2012). *The handbook of meteorology*. Scarecrow Press.
16. *LI-7500RS Open Path CO2/H2O Gas Analyzer Instruction Manual.* (2016, October). Accesado 22 de Febrero 2018, de https://www.licor.com/documents/c7tyf0czqn9ezkq1ki3b
17. Zhan, Z., Fu, Y., Yang, R. J., Xi, Z., & Shi, L. (2012). A Bayesian inference based model interpolation and extrapolation. *SAE International journal of materials and manufacturing*, *5*(2012-01-0223), 357-364.
18. Mitas, L., & Mitasova, H. (1999). Spatial interpolation. *Geographical information systems: principles, techniques, management and applications*, *1*, 481-492.
19. Wang, L. (2017). Heterogeneous Data and Big Data Analytics. *Automatic Control and Information Sciences*, *3*(1), 8-15.
20. Kruchten, P. B. (1995). The 4+ 1 view model of architecture. *IEEE software*, *12*(6), 42-50.
21. Bakken, D. (2001). Middleware. *Encyclopedia of Distributed Computing*, *11*.