PFM

ESTADO M30

SERGIO CONDE

1. PROBLEMA A RESOLVER

Se disponen de datos históricos y tiempo real con la información del tráfico e incidencias de la M30 de Madrid.

Para información de tráfico se dispone de los siguientes dataset:

- **Históricos de usuarios que circulan desde 2013** (actualización diaria), fichero en formato XML que contiene los datos totales del día de usuarios, vehículos por kilómetro en total y en los ramales, velocidad media, distancia media recorrida y tiempo medio de recorrido.
- **Estados de tráfico con coordenadas** (Actualización cada 5 minutos), fichero en formato XML que contiene las coordenadas y tipo de ocupación por color, siguiendo el siguiente patrón:
    - Tráfico fluido o poco condicionado: El archivo aparece vacío.
    - Si existen tramos con tráfico lento (obtenido cuando los datos de ocupación están entre el 21 y el 35%, lo que corresponde aproximadamente a unas velocidades de entre 25 y 55 Km/h), se muestra las coordenadas de los puntos y el color Rojo (en hexadecimal)
    - Si existen tramos con tráfico congestionado (obtenido cuando los datos de ocupación son > 35%, lo que corresponde aproximadamente a unas velocidades < 25 Km/h), se muestra las coordenadas de los puntos y el color Negro (en hexadecimal)
- **Histórico de Incidencias y accidentes** (Año 2016 y 2017), ficheros en formato CSV que contienen un amplio número de datos que interfieren en una incidencia o accidente, para el presente proyecto se ha optado por seleccionar los siguientes:
    - Fecha, hora, calzada (EXTERIOR O INTERIOR), enlace (Superficie O TUNEL), tipo accidente, tipo de colisión, número de vehículos, turismos, motocicletas y vehículos pesados implicados, el número de heridos leves, heridos graves y victimas mortales, daños en mobiliario, estado de la circulación (fluida...), estado del firme (seco...), corte de carril y una breve descripción
- **Incidencias en tiempo real** (actualizado cada 5 minutos), fichero en formato XML que contiene las coordenadas, código de incidencia, descripción y fecha de cierre de la incidencia.

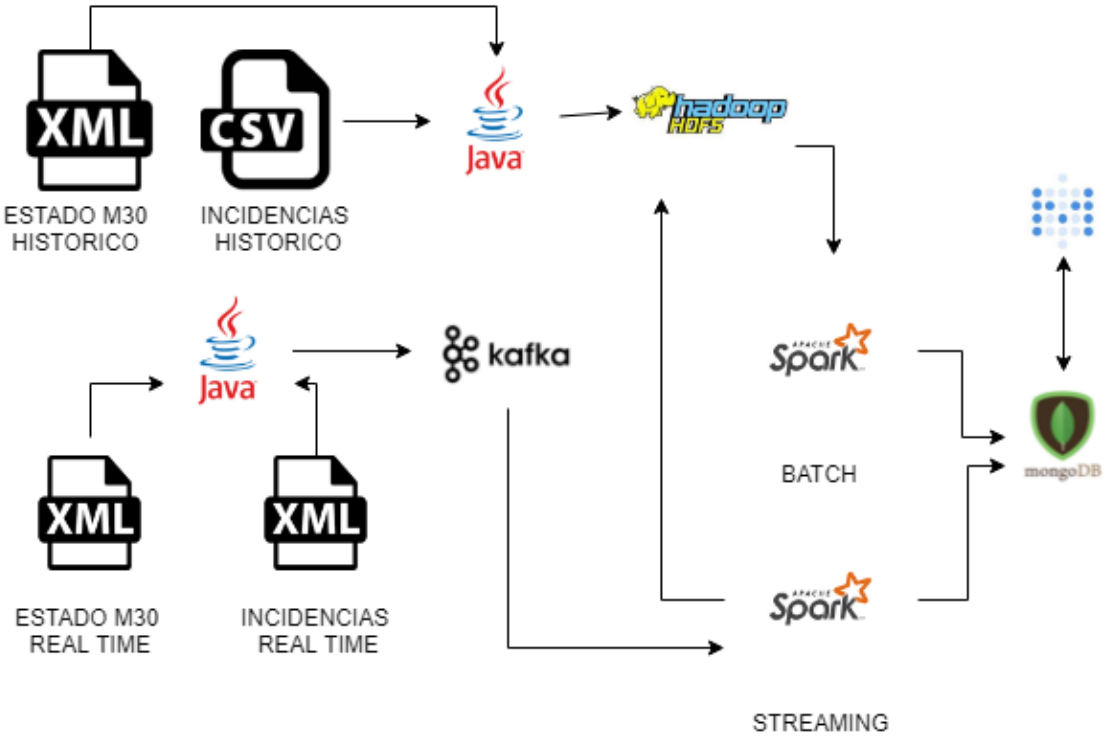
Con el presente proyecto se ha pretendido disponer de un sistema que permita realizar un control en tiempo real del estado del tráfico de la M30, y que a su vez permita, consultar la información de datos históricos.

2. SOLUCIÓN

La solución propuesta dispone de diferentes procesos para el procesamiento de cada fichero.

En primer lugar, vamos a realizar una vista general de la arquitectura utiliza para el desarrollo de la solución.

2.1. Diagrama de Arquitectura



Los dataset son proporcionados al sistema por diferentes métodos dependiendo del proceso que se trate. En el sistema se pueden diferenciar los siguientes procesamientos:

- Histórico de Estado de la M30
- Histórico de Incidencias
- Estado de la M30 en tiempo real
- Incidencias en tiempo real

Por el tipo y forma de procesamiento la inyección de documentos se hacen de diferentes formas:

- Los dataset de históricos se almacenan en HDFS, para ser procesados posteriormente en Spark, para ello se obtienen de un directorio local (histórico de incidencias) o de una url (histórico de estados) y se almacenan en HDFS en formato parquet, posteriormente se almacenan en base de datos MongoDB.
- Los dataset de tiempo real son obtenidos por medio de una url en formato XML y son volcados a una cola Kafka, que es consumida por medio de Spark Streaming, donde se almacena en HDFS en formato parquet y posteriormente se almacenan en base de datos MongoDB.

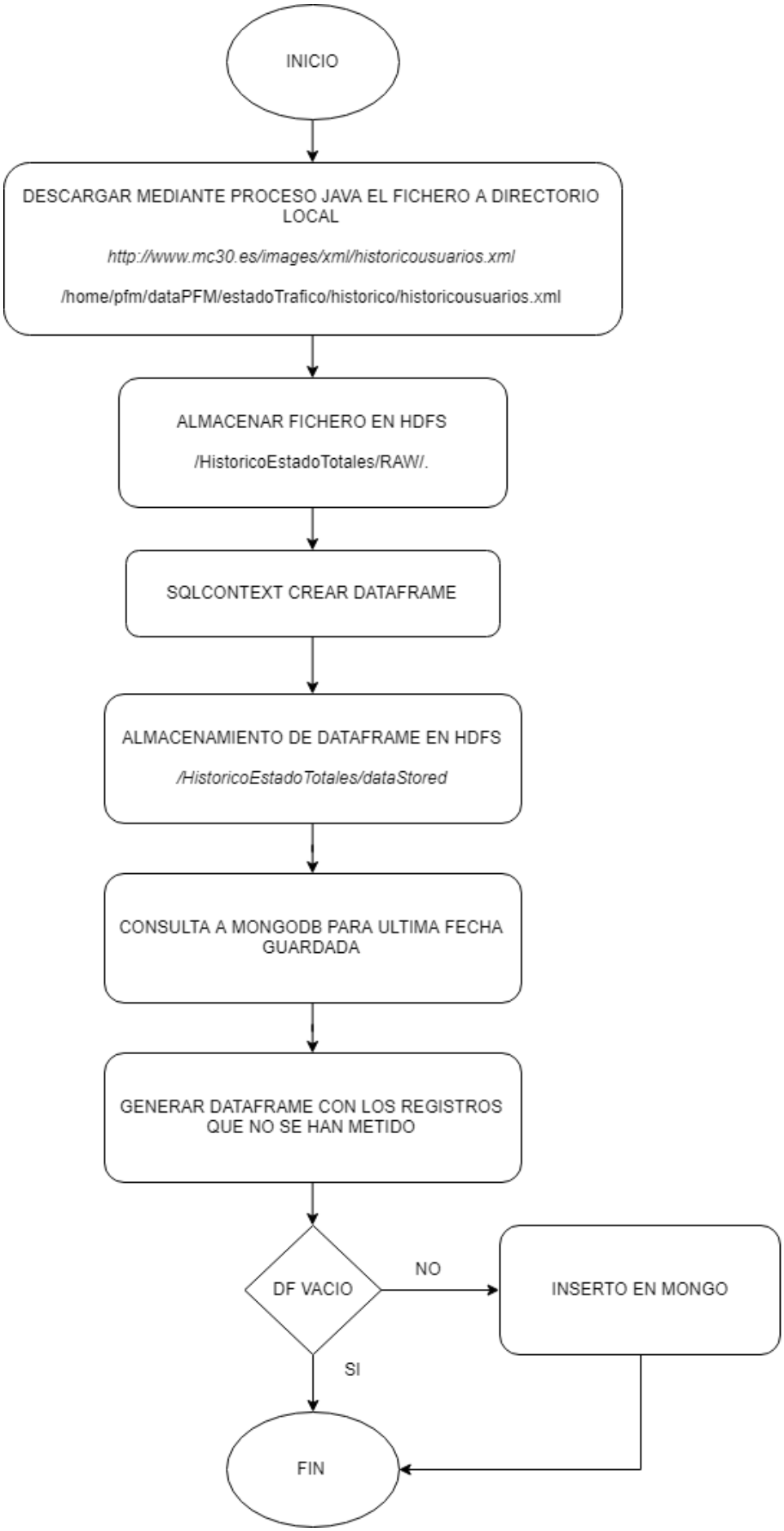
Tras el procesamiento de los datos y su almacenamiento en MongoDB, se hace uso de la herramienta Metabase para su explotación.

2.2. Diagramas de estados

2.2.1. Histórico de Estados M30

Para el procesamiento del dataset de Histórico de estados de la M30, se obtiene por medio de descarga de una URL, que nos proporciona un fichero en formato XML. Para ello, periódicamente todos los días se lanza el proceso de carga de históricos de estado *EstadoTraficoHistoricoBatch.scala* que es el encargado de llamar a la clase *HistoricosEntryPoint.java* para que acceda a la descarga del fichero en la URL *http://www.mc30.es/images/xml/historicousuarios.xml* y lo deje en la ruta local /home/pfm/dataPFM/estadoTrafico/historico/historicousuarios.xml y posteriormente lo almacene en HDFS en la ruta /HistoricoEstadoTotales/RAW/.

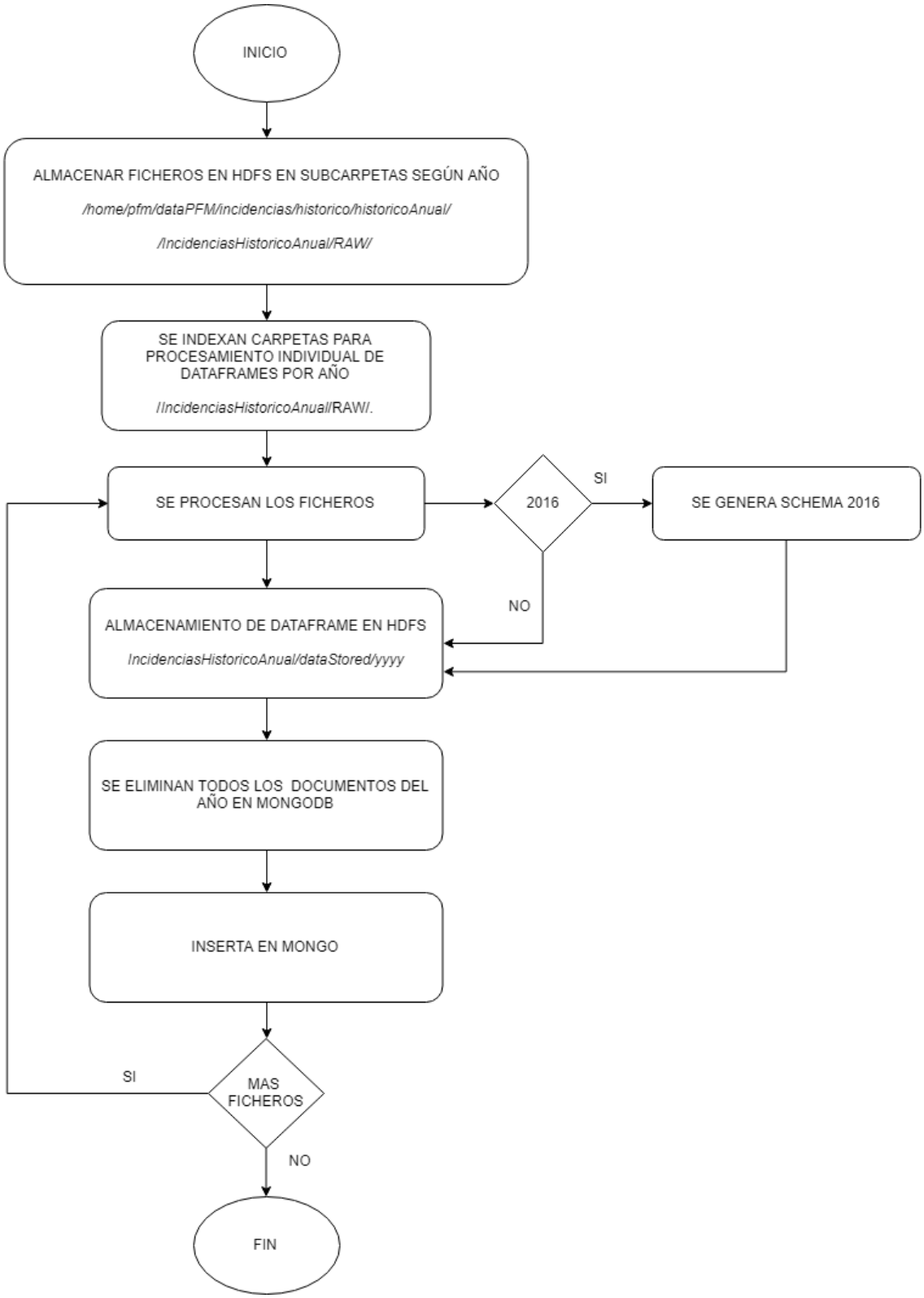
Por medio de *SQLContext* se obtiene el DataFrame del dataset y se comienza su procesamiento, en primer lugar, se almacena el dataframe en HDFS sobrescribiendo lo anterior, debido a que almacenamos el dataframe completo y el fichero contiene toda la información histórica, de modo que en caso contrario estaríamos almacenando información repetida. El almacenamiento se realiza en *parquet* en la ruta */HistoricoEstadoTotales/dataStored*. Tras almacenarlo en HDFS, obtenemos de base de datos cual fue la última fecha guardada en base de datos, para evitar insertar duplicados en la misma, con esta información se obtiene el DataFrame mediante consulta SQL con SQLContext, en la que solo se genera un dataframe con aquello elementos que aún no han sido insertados en base de datos en la colección *estadoTraficoTotales.* Recorremos todos los registros del dataframe, creando objetos de tipo *EstadoTraficoTotalPO* que serán insertados en MongoDB.


INICIO
DESCARGAR MEDIANTE PROCESO JAVA EL FICHERO A DIRECTORIO LOCAL
http://www.mc30.es/images/xml/historicousuarios.xml
/home/pfm/dataPFM/estadoTrafico/historico/historicousuarios.xml
ALMACENAR FICHERO EN HDFS
/HistoricoEstadoTotales/RAW/.
SQLCONTEXT CREAR DATAFRAME
ALMACENAMIENTO DE DATAFRAME EN HDFS
/HistoricoEstadoTotales/dataStored
CONSULTA A MONGODB PARA ULTIMA FECHA GUARDADA
GENERAR DATAFRAME CON LOS REGISTROS QUE NO SE HAN METIDO
DF VACIO
NO
INSERTO EN MONGO
SI
FIN

2.2.2. Histórico Incidencias

Para el procesamiento del dataset de Histórico de Incidencias, tiene que dejarse los ficheros con los históricos anuales en la ruta */home/pfm/dataPFM/incidencias/historico/historicoAnual/*. Para comenzar el procesamiento, se debe lanzar manualmente cuando haya terminado el año y se haya generado el histórico del año anterior, el proceso de carga de históricos de estado *IncidenciasHistoricoAnualBatch.scala* que es el encargado de llamar a la clase *HistoricoIncidenciasAnualHelper.java,* que accederá a la ruta indicada y copiará en HDFS los ficheros encontrados en la ruta indicada. El almacenamiento en HDFS se realiza en la ruta */IncidenciasHistoricoAnual/RAW/* y serán particionados por el año.

Una vez volcados los ficheros, en *IncidenciasHistoricoAnualBatch.scala,* se accede a la ruta */IncidenciasHistoricoAnual/RAW/* y se indexan las subcarpetas, para crear así un Dataframe por cada año que hay que procesar. En el caso que abordamos además de que la cabecera del dataset tiene acentos, hay diferencia en la estructura del fichero para el año 2016, por lo que se procede a crear un schema general pero que será modificado en caso de que tengamos que procesar el dataset de 2016. Tras crear el schema correspondiente se obtiene, mediante SQLContext, el DataFrame del dataset y se comienza su procesamiento, en primer lugar, se almacena en HDFS en la partición correspondiente a su año, sobrescribiendo lo anterior. El almacenamiento se realiza en formato *parquet* en la ruta *IncidenciasHistoricoAnual/dataStored.* Tras almacenarlo en HDFS, se obtiene el DataFrame mediante consulta SQL con SQLContext, como en este caso vamos a procesar el fichero del año entero que ha sido metido a mano, se entiende que por algún motivo se quiere modificar, de modo que se elimina de MongoDB los registros referentes a ese año. Por cada registro se genera un objeto de tipo IncidenciaHistoricoAnual y se almacena en base de datos.

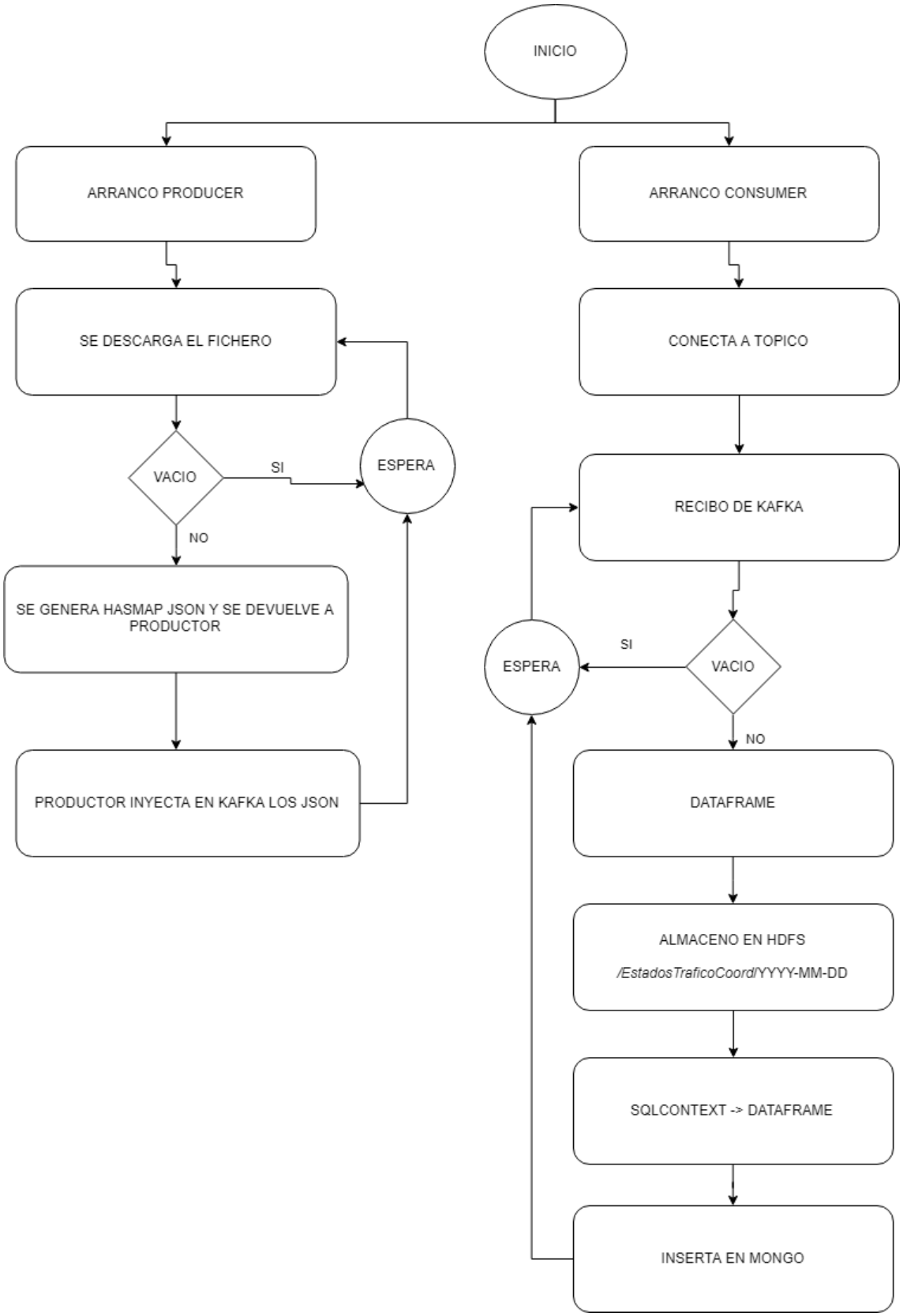

INICIO
ALMACENAR FICHEROS EN HDFS EN SUBCARPETAS SEGÚN AÑO
/home/pfm/dataPFM/incidencias/historico/historicoAnual/
/IncidenciasHistoricoAnual/RAW/
SE INDEXAN CARPETAS PARA PROCESAMIENTO INDIVIDUAL DE DATAFRAMES POR AÑO
/IncidenciasHistoricoAnual/RAW/.
SE PROCESAN LOS FICHEROS
2016
SI
SE GENERA SCHEMA 2016
NO
ALMACENAMIENTO DE DATAFRAME EN HDFS
IncidenciasHistoricoAnual/dataStored/yyyy
SE ELIMINAN TODOS LOS DOCUMENTOS DEL AÑO EN MONGODB
INSERTA EN MONGO
MAS FICHEROS
SI
NO
FIN

2.2.3. Estado de Tráfico Tiempo Real

Para el procesamiento del dataset de tiempo real del estado del tráfico en la M30, se realiza mediante procesamiento en streaming con Spark y Kafka. De modo que tenemos un productor y un consumidor. En primer lugar, el productor se ejecuta desde *EstadoTraficoProducer.scala*, que está continuamente en ejecución. En el arranque y en cada iteración, hace una petición de inyección de datos a *EstadosTraficoXMLParser.java,* encargada de obtener el fichero con el dataset descargándolo a través de la URL http://www.mc30.es/components/com_hotspots/datos/estado_trafico.xml tras la descarga formatea el XML de tal forma que devuelve un HashMap con todos los registros en formato JSON válidos para la Spark, en caso de no contener información el parser devuelve nulo.

Por su parte, el productor obtiene el HashMap y extrae los JSON contenidos en el mismo y los inyecta al tópico *EstadoTrafico*, tras lo cual se queda a la espera, hasta alcanzar el tiempo de la siguiente iteración.

El consumer se ejecuta desde *EstadoTraficoConsumer.scala* y se conecta al tópico *EstadoTráfico.scala* mediante un *DirectStream*k de Kafka, a través del cual obtiene un RDD por cada JSON enviado desde el lado del productor, que en el lado del consumidor es convertido a Dataframe mediante SQLContext y se almacena el en HDFS añadiéndose en /*EstadosTraficoCoord*/YYYY-MM-DD, a lo que hubiese anteriormente en la partición correspondiente a la fecha, este almacenamiento se realiza en *parquet*. Tras almacenarlo en HDFS, mediante consulta SQL con SQLContext, se genera un dataframe que recorreremos e iremos insertando en la colección *estadoTraficoCoord* mediante la creación del objeto *EstadoTrafico.*

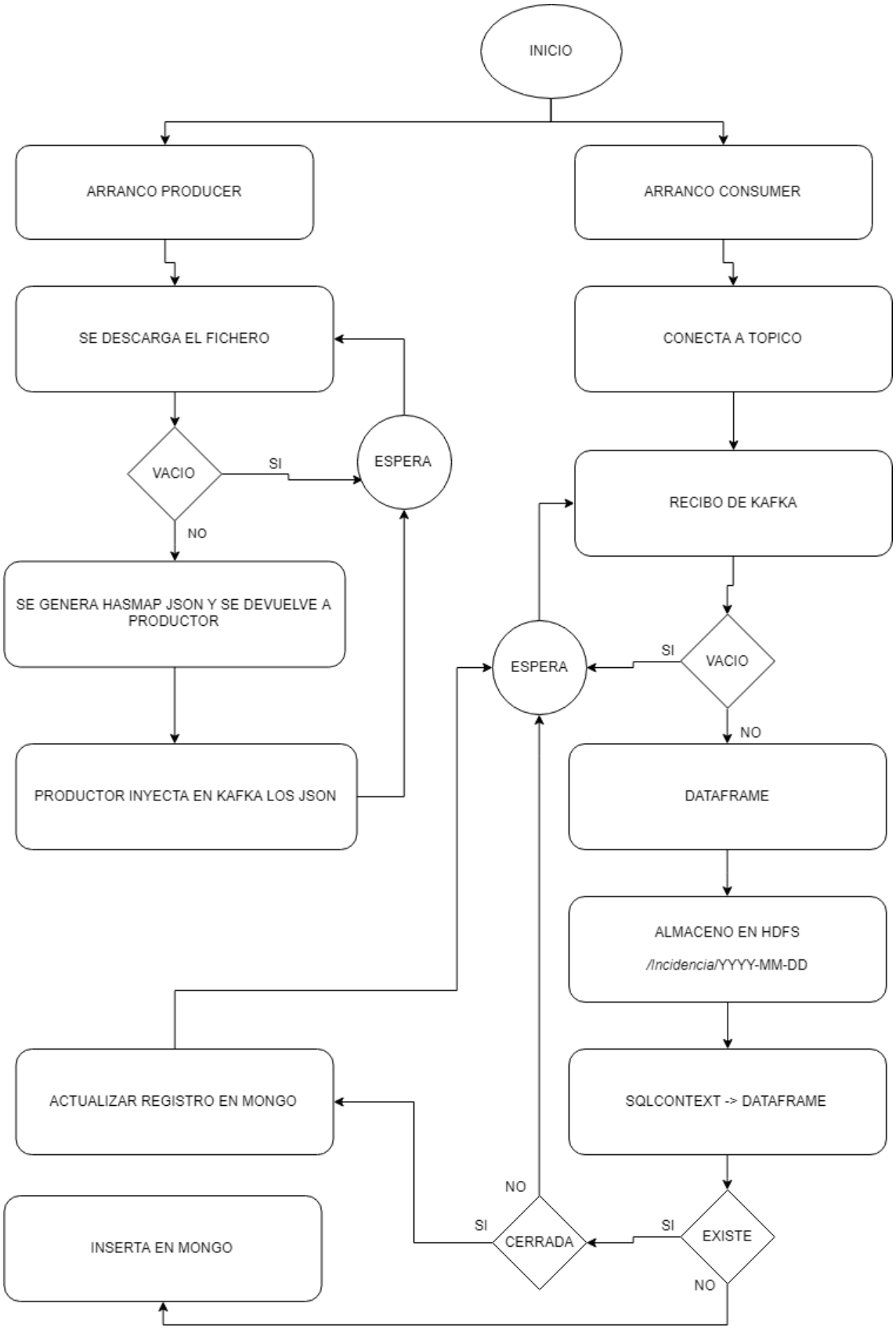

INICIO
ARRANCO PRODUCER
ARRANCO CONSUMER
SE DESCARGA EL FICHERO
CONECTA A TOPICO
VACIO
SI
ESPERA
NO
RECIBO DE KAFKA
SE GENERA HASMAP JSON Y SE DEVUELVE A PRODUCTOR
ESPERA
SI
VACIO
NO
PRODUCTOR INYECTA EN KAFKA LOS JSON
DATAFRAME
ALMACENO EN HDFS
/EstadosTraficoCoord/YYYY-MM-DD
SQLCONTEXT -> DATAFRAME
INSERTA EN MONGO

2.2.4 Estado de Incidencias Tiempo Real

Para el procesamiento del dataset de tiempo real de incidencias se realiza mediante procesamiento en streaming con Spark y Kafka. De modo que tenemos un productor y un consumidor. En primer lugar, el productor se ejecuta desde *IncidenciasTRProducer.scala*, que está continuamente en ejecución. En el arranque y en cada iteración, hace una petición de inyección de datos a *IncidenciasXMLParser.java,* encargada de obtener el fichero con el dataset descargándolo a través de la URL http://www.mc30.es/components/com_hotspots/datos/incidencias.xml tras la descarga formatea el XML de tal forma que devuelve un HashMap con todos los registros en formato JSON válidos para la Spark, en caso de no contener información el parser devuelve nulo.

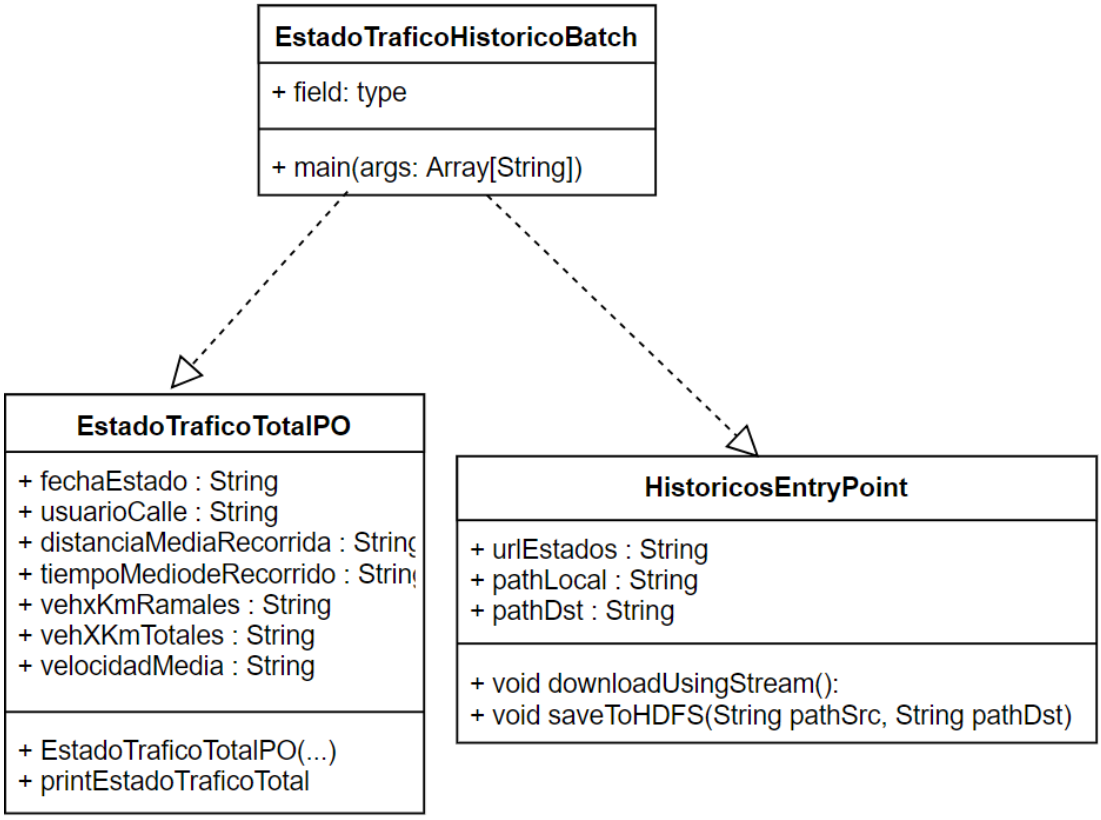
Por su parte, el productor obtiene el HashMap y extrae los JSON contenidos en el mismo y los inyecta al tópico *IncidenciasTR*, tras lo cual se queda a la espera, hasta alcanzar el tiempo de la siguiente iteración.

El consumer se ejecuta desde *EstadoTraficoConsumer.scala* y se conecta al tópico *EstadoTráfico.scala* mediante un *DirectStream*k de Kafka, a través del cual obtiene un RDD por cada JSON enviado desde el lado del productor, que en el lado del consumidor es convertido a Dataframe mediante SQLContext y se almacena el en HDFS en */Incidencia*/YYYY-MM-DD, añadiéndose a lo que hubiese anteriormente en la partición correspondiente a la fecha, este almacenamiento se realiza en *parquet*. Tras almacenarlo en HDFS, mediante consulta SQL con SQLContext, se genera un dataframe que recorreremos y tenemos que consultar en la base de datos si ya existe, para actualizarlo y evitar así la duplicación de la información. En caso de no existir se inserta en la colección *incidenciasCoord* mediante la creación del objeto *EstadoTrafico,* pero si ya se encuentra en base de datos hay que comprobar si nos llega con fecha de cierre para actualizarla y dar por cerrada la incidencia o simplemente todavía se encuentra abierta, en tal caso descartaremos la incidencia.

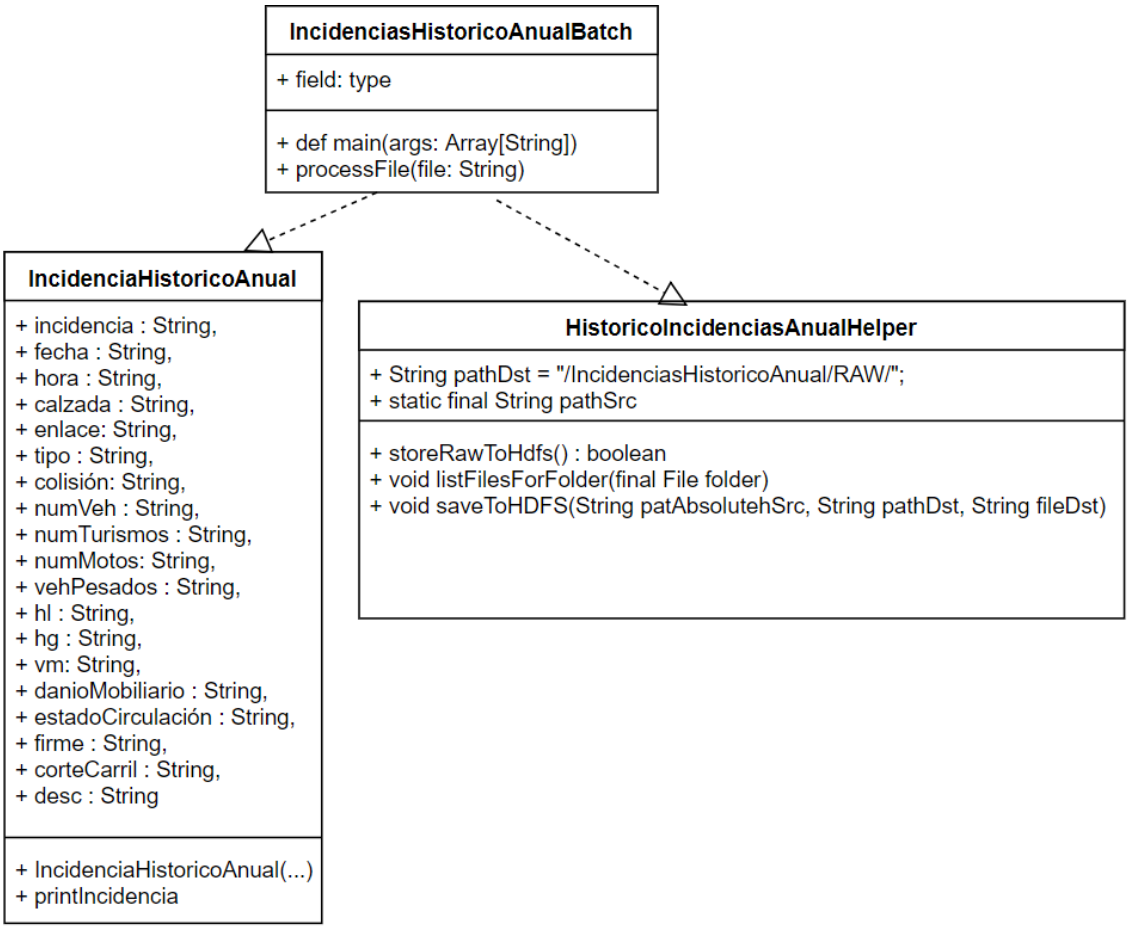

INICIO
ARRANCO PRODUCER
ARRANCO CONSUMER
SE DESCARGA EL FICHERO
CONECTA A TOPICO
VACIO
SI
ESPERA
NO
RECIBO DE KAFKA
SE GENERA HASMAP JSON Y SE DEVUELVE A PRODUCTOR
ESPERA
SI
VACIO
NO
PRODUCTOR INYECTA EN KAFKA LOS JSON
DATAFRAME
ALMACENO EN HDFS
/Incidencia/YYYY-MM-DD
ACTUALIZAR REGISTRO EN MONGO
SQLCONTEXT -> DATAFRAME
NO
SI
SI
EXISTE
INSERTA EN MONGO
CERRADA
NO

2.3. Diagrama de Clases

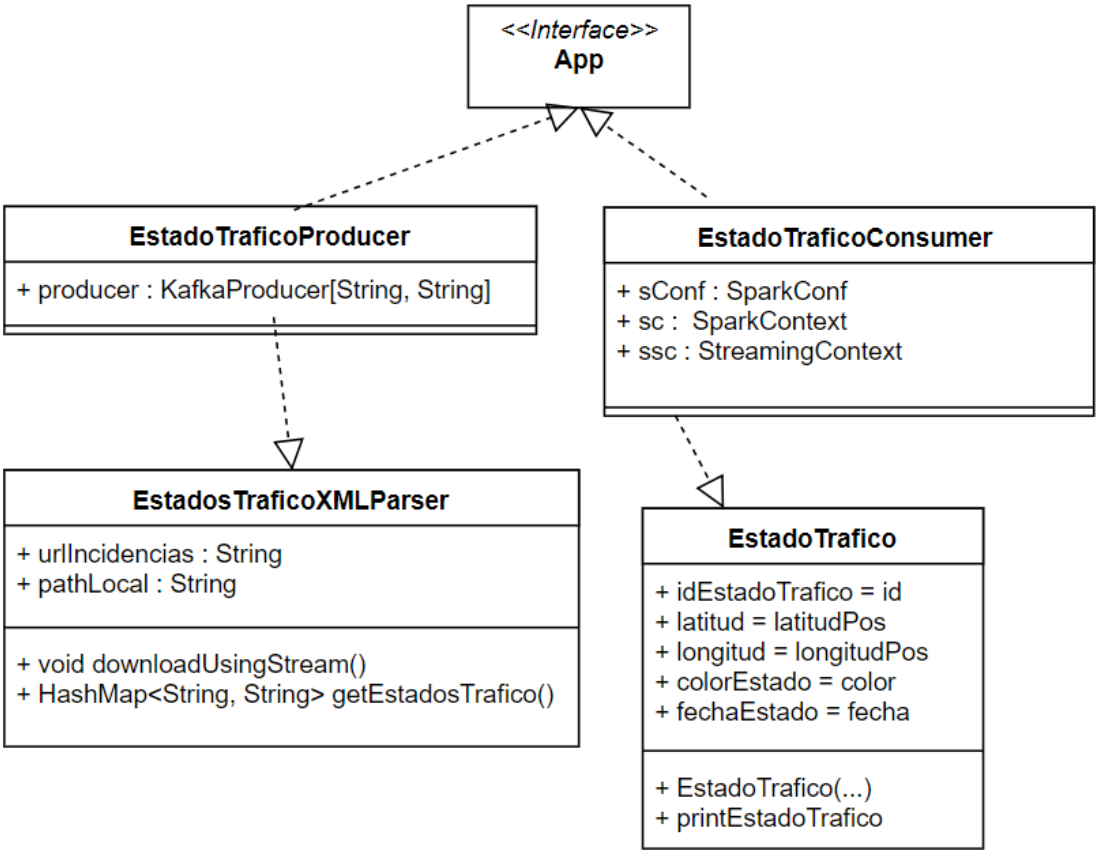
2.3.1. Histórico de Estado de Tráfico



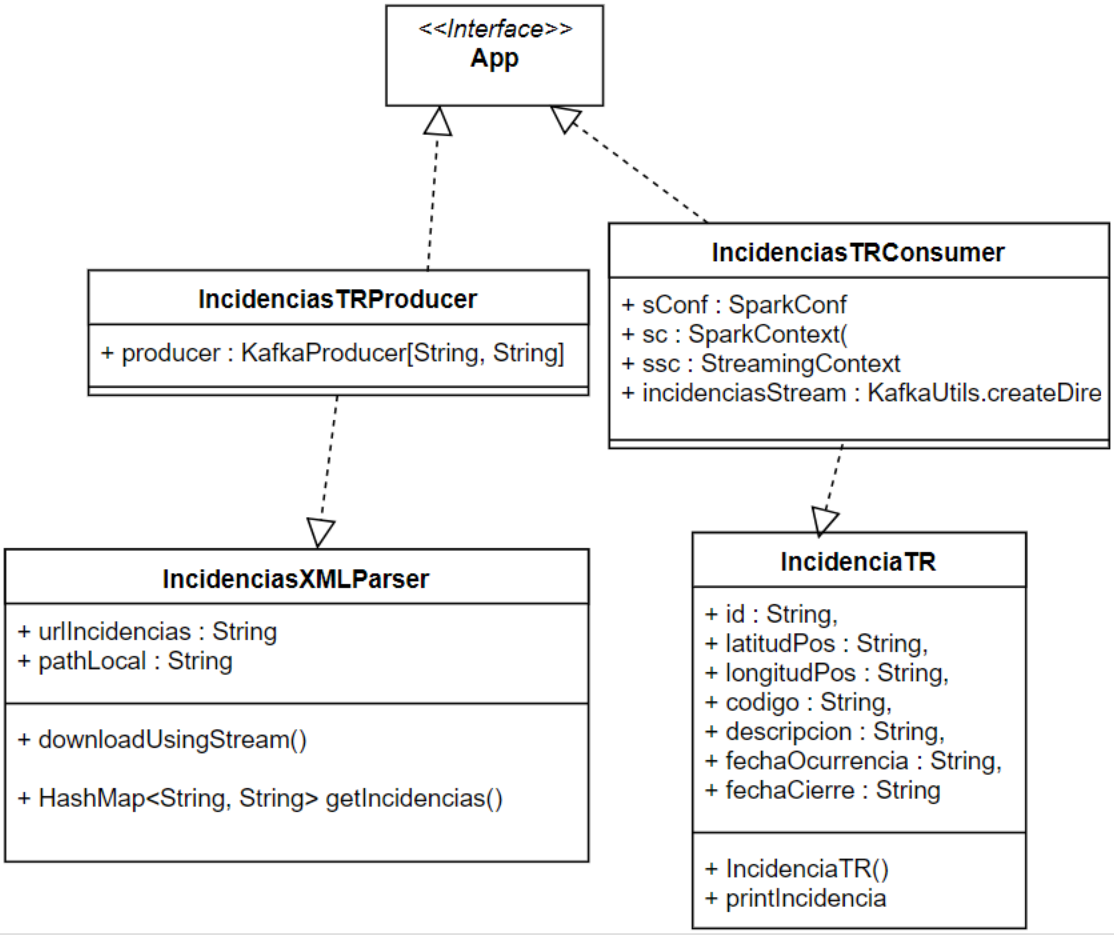
2.3.2. Histórico de Incidencias



2.3.3. Estado de tráfico en tiempo real



2.3.4. Incidencias en tiempo real



2.4. Modelo de datos

En este apartado se mostrarán como se reciben los datos y como se ha procedido a su almacenamiento en la base de datos MongoDB. Para su almacenamiento se ha creado un schema nuevo llamado *pfm* y cuatro colecciones para almacenar los documentos:

- estadoTraficoCoord
- estadoTraficoTotales
- historicoAnualIncidencias
- incidenciasCoord

2.4.1. Histórico de estado de tráfico

El fichero XML de históricos de estado de tráfico proporcionado se pueden extraer los siguientes datos:

- Fecha en formato dd/mm/aaaa
- Número total de usuarios de calle 30

- Vehículos por kilómetros totales
- Vehículos por kilómetros ramales
- Velocidad media
- Distancia Media Recorrida
- Tiempo Medio Recorrido

Los cuales has sido almacenados como String en MongoDB de siguiendo las siguientes nomenclaturas:

```
db.estadoTraficoTotales
{
    "_id" : ObjectId,
    "fecha" :,
    "usuarios" :,
    "distanciaMedia",
    "tiempoMedio",
    "vehKmRamales",
    "vehKmTotales",
    "velMedia"
}
```

2.4.2. Histórico incidencias

El fichero CSV de histórico de incidencias nos proporciona la siguiente información

- Fecha
- Hora
- Día
- Nº Incidencia
- sitrem
- Carretera
- MC30
- Calzada
- Localización
- Enlace
- Tipo de accidente
- Tipo de colisión
- Nº Vehículos implicados
- Nº Turismos

- Nº Motocicletas
- Nº vehículos pesados
- H.L. → heridos leves
- Hl → herido leves (En 2016 aparece duplicado este registro)
- Hg → heridos graves
- Vm → victimas mortales
- Daños al mobiliario urbano
- Circulación
- Estado del Firme
- Presencia Recursos Externos
- Corte Carril
- Vertido
- Grúa usuario
- Grúa Propia EMESA
- Grúa Movilidad
- Factores Atmosféricos
- Visibilidad señalización
- Señalización de Peligro
- Visibilidad restringida por
- Aceras
- Árboles
- Vehículos denominación
- Descripción del accidente
- Daños causados al viario
- Materiales utilizados
- Observaciones

En este caso se ha decidido descartar algunos campos de modo que el almacenamiento en base datos queda como sigue:

db.historicoAnualIncidencias

{

"_id" : ObjectId,

"incidenciaVI",

"fechaVI",

"horaVI",

"calzadaVI",

"enlaceVI",

"tipoVI",

"colisiónVI",

"numVehVI",

"numTurismosVI",

"numMotosVI",

"vehPesadosVI",

"hlVI",

"hgVI",

"vmVI",

"danioMobiliarioVI",

"estadoCirculaciónVI",

"firmeVI",

"corteCarrilVI",

"descVI"

}

2.4.3. Estado de tráfico en tiempo real

El fichero XML de estado de tráfico en tiempo real nos ofrece la siguiente información:

- Posición (Latitud, Longitud) --> Listado de coordenadas afectadas
- Color

Para su almacenamiento se decidió crear un id único compuesto único y almacenar la fecha actual del sistema para registrar la ocurrencia del registro

db.estadoTraficoCoord

{
"_id",
"idEstadoTrafico",
"latitud",
"longitud",
"colorEstado",
"fechaEstado"
}

2.4.4. Incidencias en tiempo real

- Posición (Latitud, Longitud)
- Identificador -> Código generado a partir de la fecha de ocurrencia de la incidencia
- Código propio

- Descripción de la incidencia ("Vehículo parado en arcén....")
- Fecha de cierre --> SIN CERRAR hasta desaparición de la incidencia

db.incidenciasCoord

{

"_id" : ObjectId("5b16e53e96063455bf78d44e"),

"idIncidencia",

"latitud",

"longitud",

"codigoIncidencia",

"desc",

"fechaIncidencia",

"fechacierreIncidencia"

}