



Sistemas distribuidos (75.74)

TP-2

COVID19 Statistics

---

Apellido y Nombre	Padrón	Correo electrónico
Funes Federico	98372	fede.funes96@gmail.com

---

# Índice

<b>1. Alcance</b>	<b>3</b>
<b>2. Representación de la arquitectura</b>	<b>3</b>
<b>3. Metas y restricciones de la arquitectura</b>	<b>3</b>
<b>4. Escenarios</b>	<b>4</b>
<b>5. Vista física</b>	<b>5</b>
5.1. Diagrama de robustez . . . . .	6
5.1.1. Ventajas y desventajas del diseño . . . . .	10
5.2. Diagrama de despliegue . . . . .	11
<b>6. Vista de desarrollo</b>	<b>12</b>
6.1. Diagrama de paquetes . . . . .	13
<b>7. Vista de procesos</b>	<b>14</b>
7.1. Diagrama de actividades . . . . .	15
7.2. Diagrama de secuencia . . . . .	17
<b>8. Vista lógica</b>	<b>18</b>
8.1. Diagrama de clases . . . . .	19
<b>9. Manual de uso</b>	<b>23</b>
<b>10. Problemas encontrados y mejoras disponibles</b>	<b>24</b>

## **1. Alcance**

Se propone diseñar una arquitectura para analizar una masiva cantidad de información respecto a casos de coronavirus en ciudades distribuidas por Italia, y, documentar y mostrar a través de distintas vistas el diseño de la arquitectura planteada para el problema.

No se van a incluir detalles a muy bajo nivel sino más bien a gran nivel porque el tipo de problema desarrollado no requiere de operaciones muy sofisticadas sino de una arquitectura bien planteada.

## **2. Representación de la arquitectura**

Este documento presenta la arquitectura diseñada siguiendo el modelo de documentación 4+1 mostrando los diferentes tipos de vistas usando UML. Se hará un fuerte enfoque en lo que respecta a la comunicación entre nodos porque, respecto a la implementación individual de cada nodo, el código utilizado no es muy extenso y no varía demasiado entre nodo. Por esta razón, no se incluyen muchos diagramas de secuencia ni de clases, los mismos suelen operar de manera similar (Las operaciones se implementaron de tal forma que todas prosiguen la misma metodología) y utilizando muy pocas clases, en muchos casos solo es necesario un trabajador por dato entrante y un controlador para la comunicación.

## **3. Metas y restricciones de la arquitectura**

Se propone diseñar una arquitectura para analizar una masiva cantidad de información respecto a casos de coronavirus en ciudades distribuidas por Italia. Esta arquitectura deberá cumplir con:

- Permitir escalar manualmente para soportar más carga de datos.
- Coordinar varios procesos para realizar un análisis por lote de datos y no obtener resultados parciales.
- Distribuir bien las responsabilidades de cada nodo y ser flexible.

Sin embargo, se poseen las siguientes restricciones que afectan directamente la arquitectura:

- Ciertas operaciones requieren realizarse en un único nodo (Esto serían los agrupamientos por ciudad y por fecha por ejemplo)
- Para evitar mostrar resultados parciales, es necesario tener algún nodo o nodos que controlen como manejar cuando se termina un lote de datos.

## 4. Escenarios

En el siguiente diagrama de casos de uso se puede ver como se utiliza el sistema:

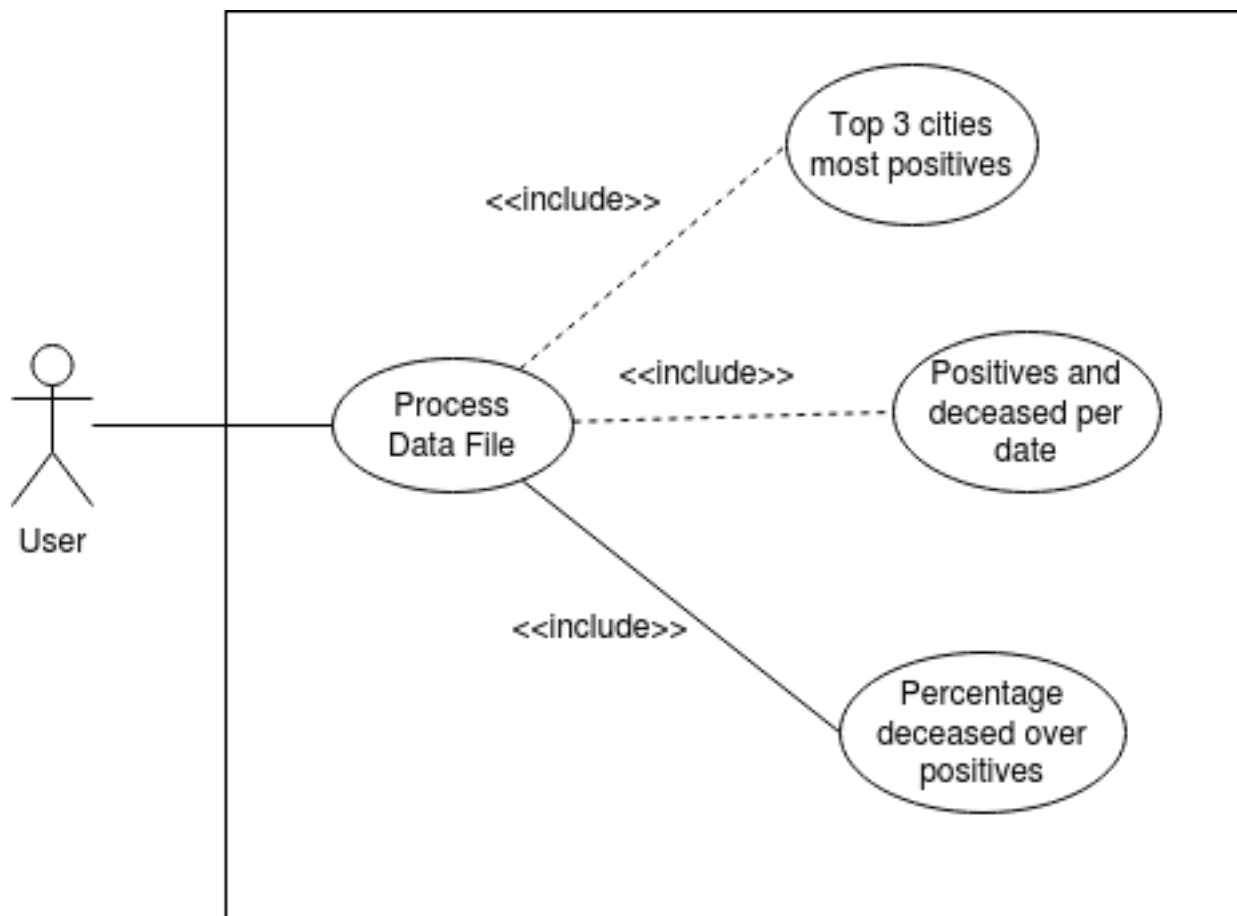


Figura 1: Casos de uso

En este caso hay un único flujo: El usuario/actor indica que comience el procesamiento del archivo y el sistema escribe en un archivo la información procesada, incluye las 3 ciudades con más casos positivos, la cantidad de positivos y decesos por fecha y el porcentaje de decesos respecto a positivos.

## 5. Vista física

Se dispone a mostrar la arquitectura del sistema (Robustez) y la distribución en nodos físicos (despliegue) mostrando ventajas y desventajas de la arquitectura definida.

## 5.1. Diagrama de robustez

Se tiene una arquitectura definida de la siguiente manera:

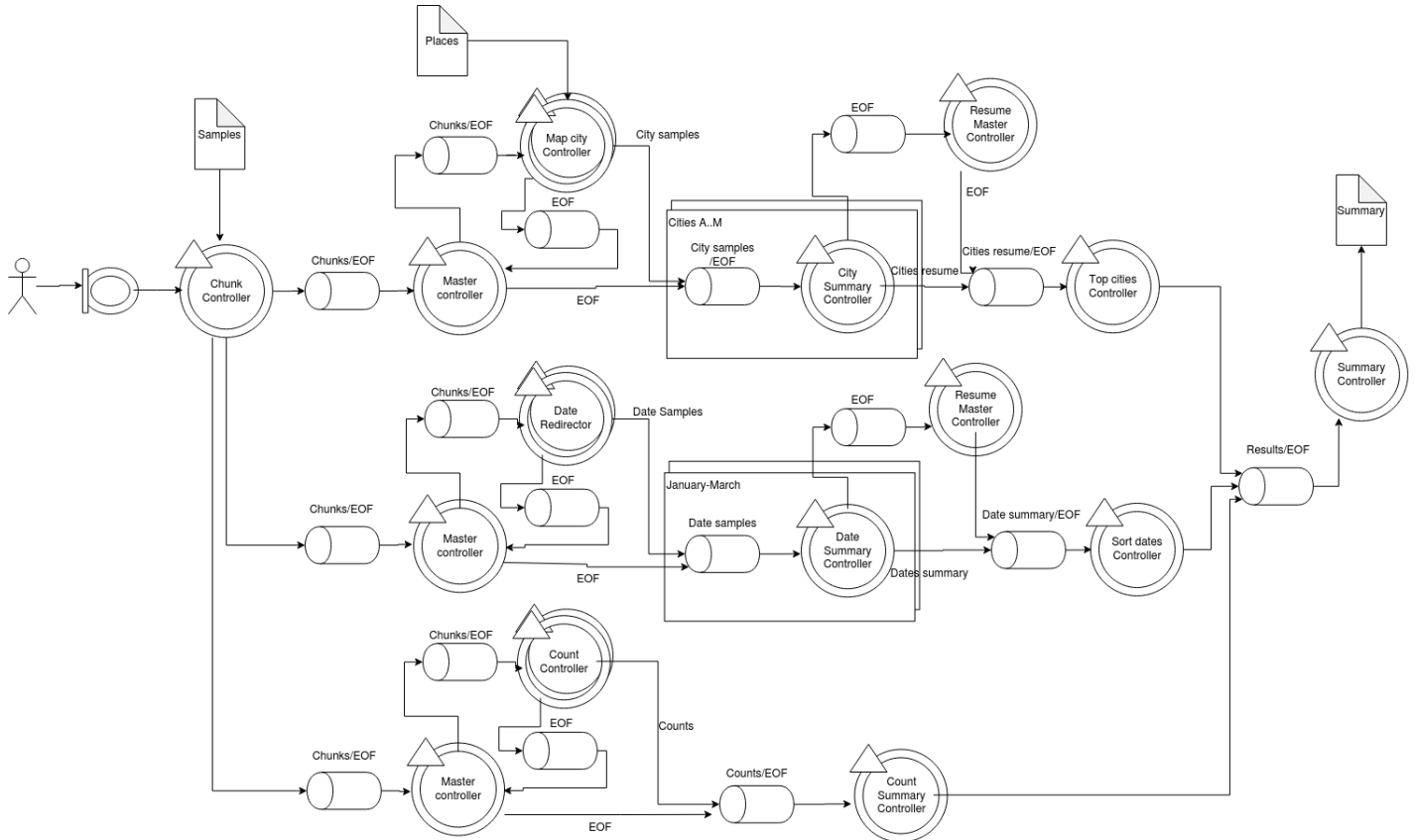


Figura 2: Diagrama de robustez completo

En la cual se encuentran distribuidas las responsabilidades de la siguiente forma:

- **Chunk manager:** Se encarga de leer del archivo de datos (Samples) y enviar información de a partes a cada cola, cada una correspondiendo a una operación diferente (Top 3 ciudades con más positivos, agrupamiento de casos por fecha y porcentaje de decesos con respecto a positivos).

- Master Controller: Se encarga de propagar información a sus correspondientes trabajadores (Workers) y de transmitir correctamente el fin de archivo (EOF) recibido del Chunk Manager para finalizar el trabajo de los Workers y propagar el fin de archivo (EOF) a los siguientes procesos.
- Map City Controller: Se encarga de calcular, filtrar y establecer la ciudad más cercana a la latitud y longitud de los datos recibidos para luego propagarlo a la cola correspondiente según ciudad.
- Date Redirector: Se encarga de transformar y redirigir los datos a la cola correspondiente según la fecha.
- Count Controller: Se encarga de contar ocurrencias de resultados (positivos o negativos) de los datos recibidos y propagar los resultados una vez recibido el fin de archivo.
- City Summary Controller: Se encarga de agrupar positivos por ciudad y propagarlos a la siguiente cola. A su vez, propaga el fin de archivo (EOF) a un controlador maestro que se encarga de pasarlo a la siguiente etapa.
- Resume Master Controller: Se encarga de recibir los fin de archivos de varios nodos y propagar un único fin de archivo cuando reciba todos los fin de archivo esperados. Es el controlador que maneja el fin de archivo.
- Date Summary Controller: Se encarga de agrupar positivos y negativos por fecha y propagarlos a la siguiente cola. También, propaga el fin de archivo a su Resume Master Controller.

- Count Summary Controller: Recibe resultados parciales de los trabajadores (Count Controllers) y calcula el resultado final agrupando la información recibida, finalmente la transmite a la cola de salida.
- Top Cities Controller: Recibe los resultados de agrupamiento por ciudad de cada controlador y calcula el top 3 con los resultados obtenidos, procede a enviarlos a la cola de salida.
- Sort Dates Controller: Recibe los resultados de agrupamiento por fecha, los ordena y los reenvía a la cola de salida.
- Summary Controller: Recibe toda la información y se encarga de escribir en un archivo los resultados obtenidos de cada operación.

Este diseño se ve bastante complejo, para verlo de manera resumida (abusando un poco de la notación utilizada) podemos hacer los siguientes reemplazos:



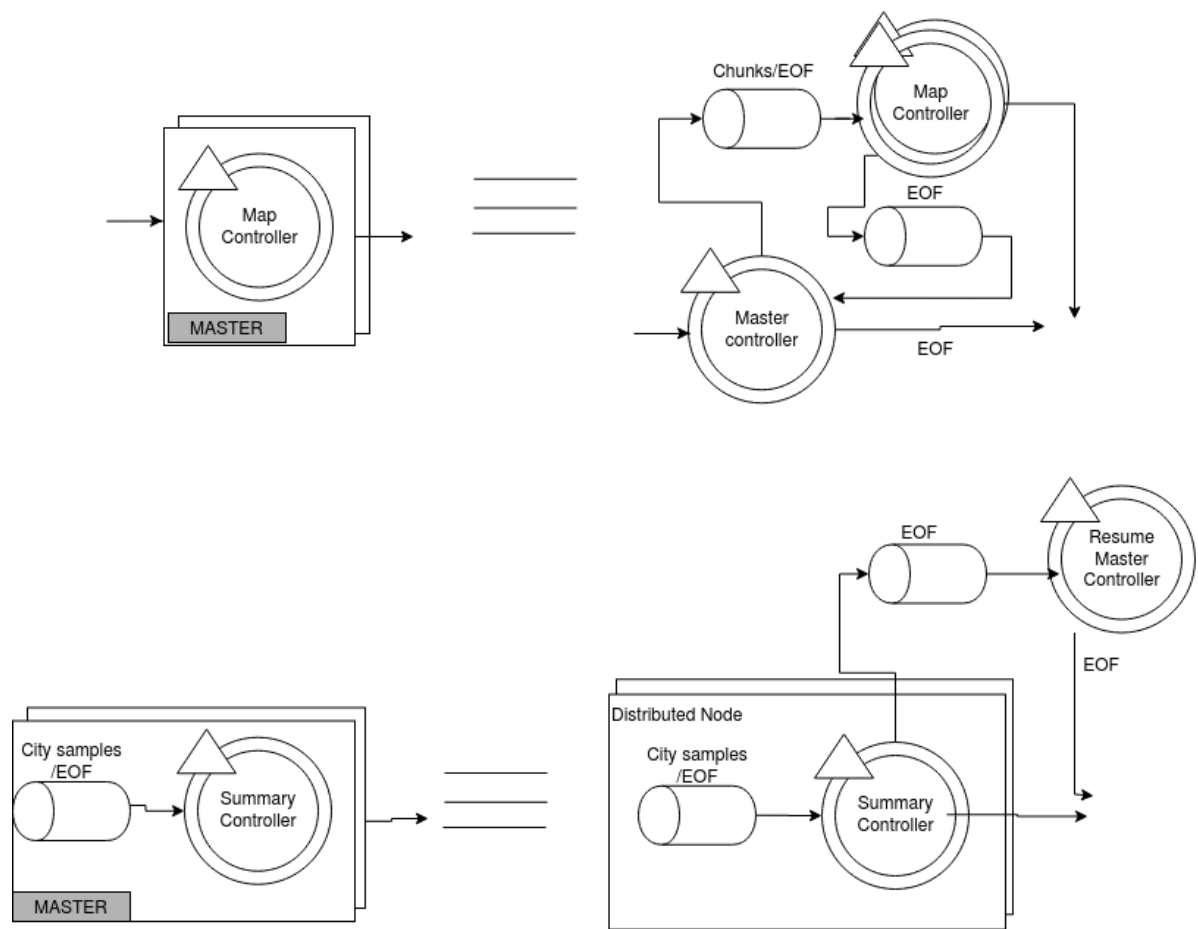


Figura 3: Propuesta de reducción

Obtenemos el siguiente diagrama de robustez reducido:

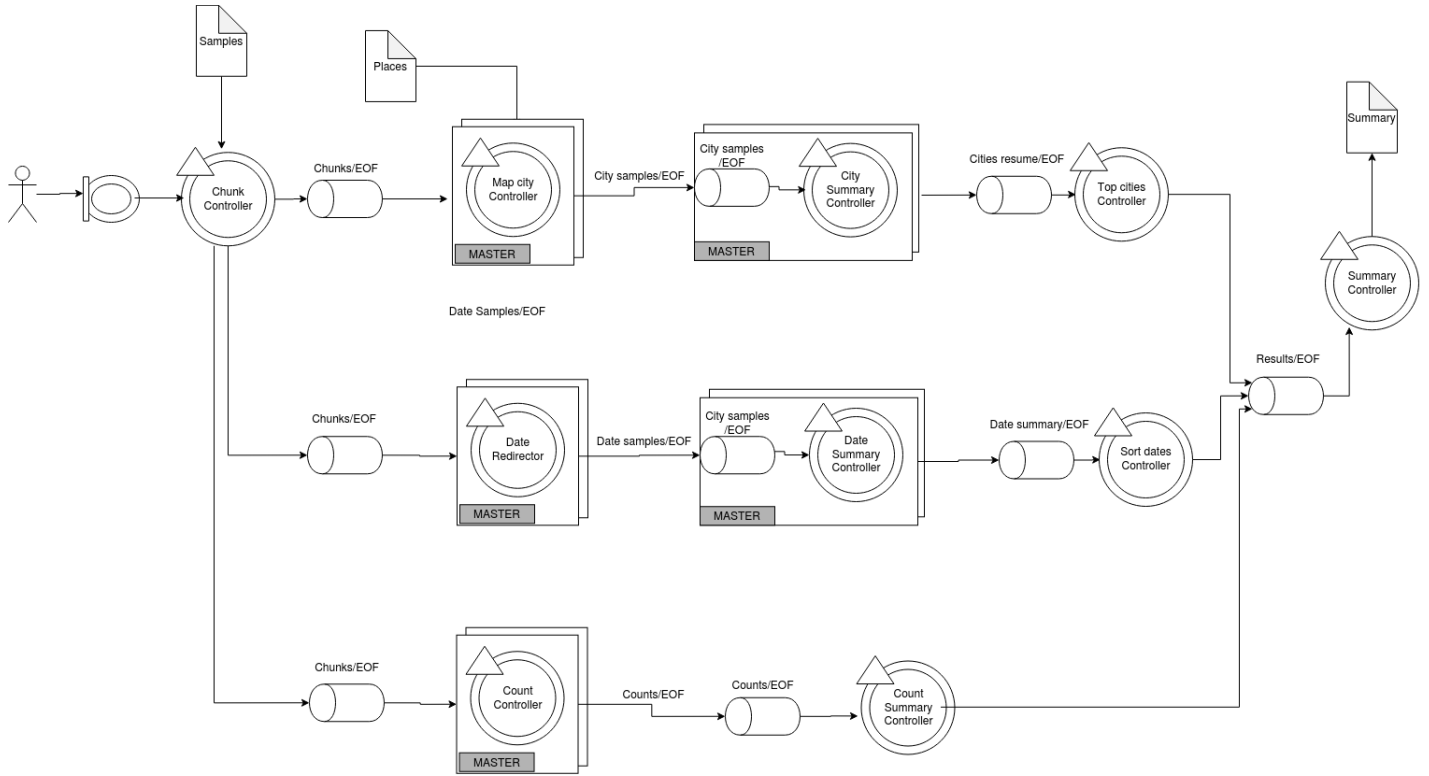


Figura 4: Diagrama de robustez reducido

Del cual es mucho más fácil ver que las 3 operaciones utilizan controladores en común que podrían abstraerse y que son caminos bastante similares. Podemos ver como ventaja que las operaciones de transformación y redirección de datos (Map y Redirect) y las operaciones de agrupamiento (Resume/Summary) pueden generalizarse.

#### 5.1.1. Ventajas y desventajas del diseño

Poseemos las siguientes ventajas:

- Agregar nuevas operaciones implica agregar un camino diferente que no requiere modificación a grandes rasgos de la arquitectura definida.
- Podemos aprovechar las generalizaciones de las operaciones menciona-

das para realizar otros tipos de análisis con la información dada.

- Permite una buena distribución de trabajo (Múltiples trabajadores) para realizar operaciones costosas de transformación de datos.
- Permite una buena distribución del volumen de datos, a fin de cuentas los nodos que poseen operaciones de agrupamiento (Resume/Summary) terminan dividiendo los datos de entrada según la cantidad de nodos que se quieran utilizar.

Sin embargo, tenemos también desventajas, como las siguientes:

- Tenemos varios puntos importantes de falla, algunos provocados por la propia naturaleza del problema (Ciertas operaciones requieren el agrupamiento de todos los datos) y otros generados por el uso de nodos maestro que controlen a los diferentes trabajadores.
- Hay más costo de transmisión de datos, el hecho de que el Chunk Manager transmita información replicada a través de 3 colas implica más costo de transporte de datos.

## 5.2. Diagrama de despliegue

Hay múltiples propuestas para esto, dado el diagrama de robustez podemos aplicar distintos cortes, entre ellos uno podría ser ubicar todos los controladores maestros en un único nodo o también ubicar todos los controladores de trabajo (Workers) en un único nodo, sin embargo, se propone la siguiente forma de desplegar:

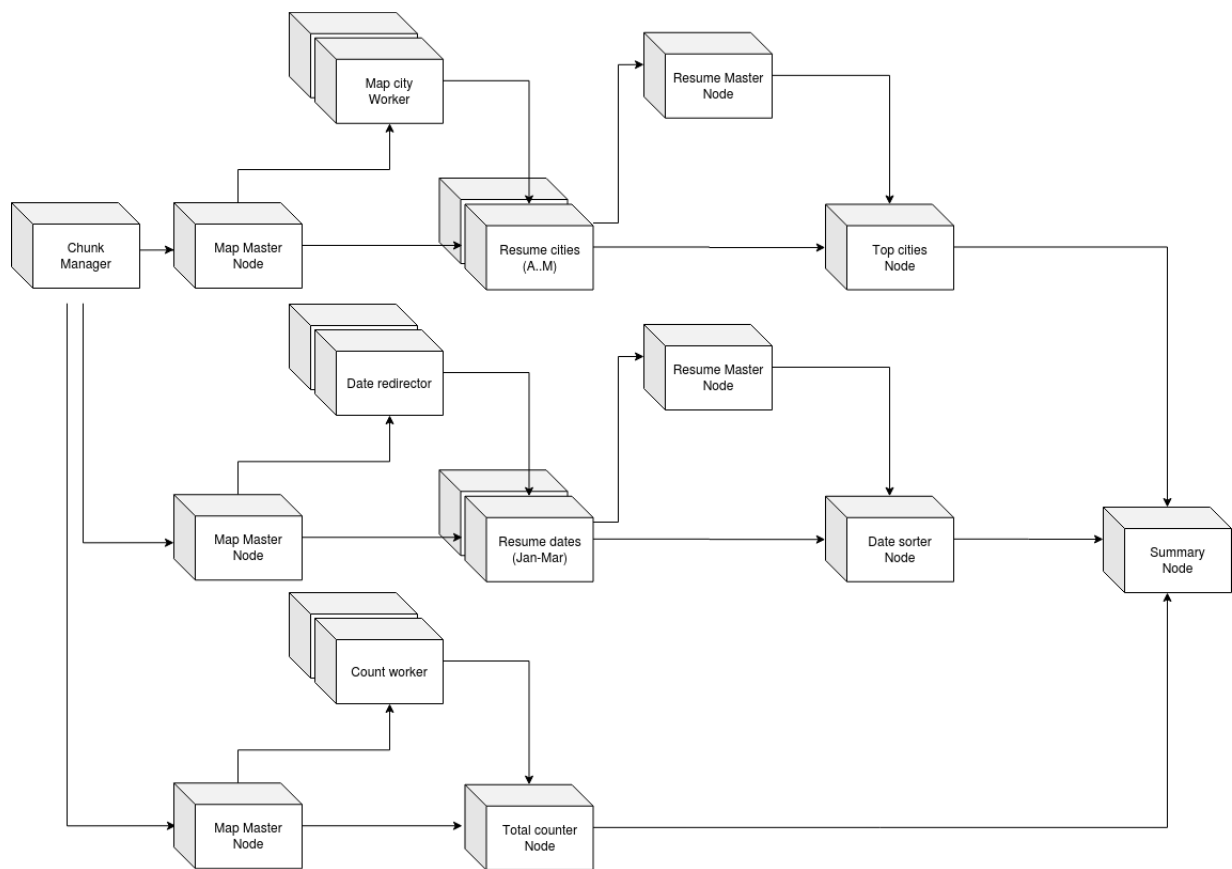


Figura 5: Diagrama de despliegue

Esta forma permite que cada nodo efectúe una única o muy pocas operaciones. Pero igual que antes, vemos nodos que son críticos y ante una caída provocarían la caída total del sistema.

## 6. Vista de desarrollo

A continuación, se van a mostrar los artefactos que componen al sistema, en este caso se optó por utilizar un diagrama de paquetes:

## 6.1. Diagrama de paquetes

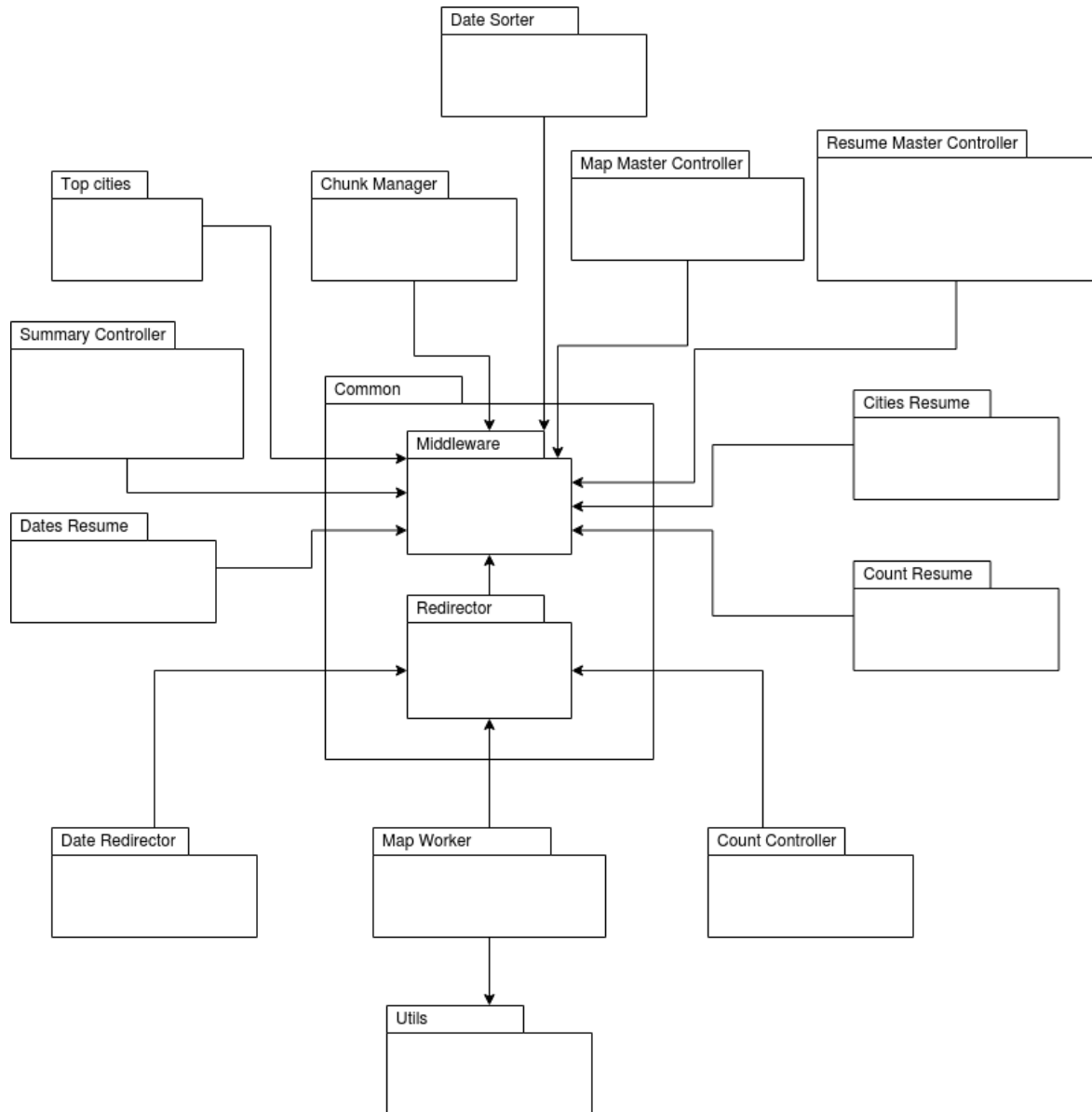


Figura 6: Diagrama de paquetes

En este caso, el paquete de Middleware es muy utilizado porque es el que permite la comunicación entre los nodos, así como también el paquete Redirector que se encarga de redirigir información a varios nodos.

## **7. Vista de procesos**

A continuación, se van a mostrar diagramas de actividades y secuencia para el flujo de las operaciones del calculo de Top 3 de ciudades con más positivos, para el resto de casos no se realiza ya que son similares los flujos (Esto es algo que ya se explicó en la Vista física - Diagrama de Robustez)

## 7.1. Diagrama de actividades

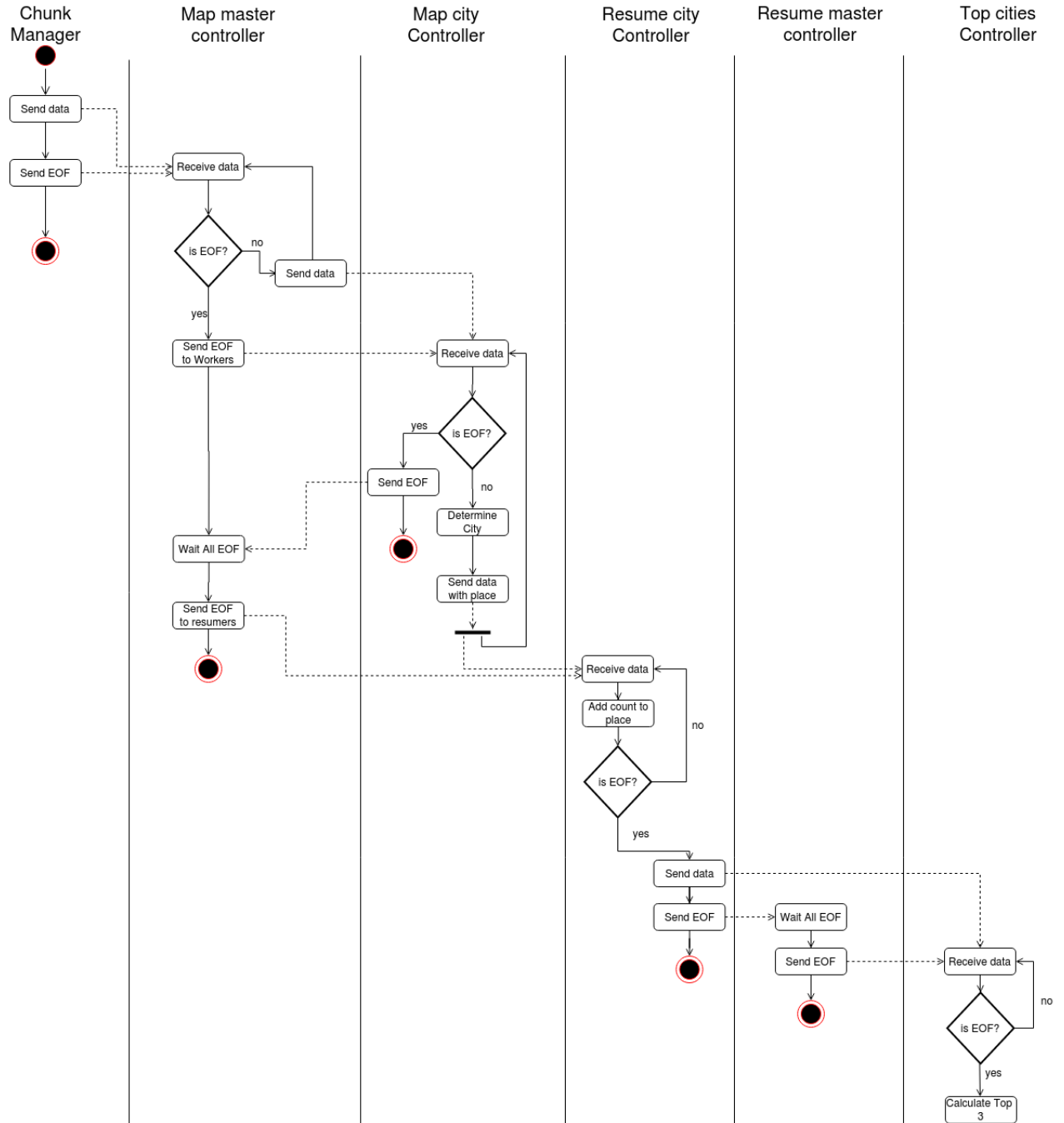


Figura 7: Diagrama de actividades del calculo de Top 3 ciudades. Parte 1/2

En esta primera parte se ve el flujo de la información hasta el calculo del

Top 3 de ciudades con más casos positivos. Algunos comentarios al respecto:

- Respecto a "Send Data" del Chunk Manager: Esta operación implica un bucle infinito de enviar información hasta llegar al fin de archivo, no se impuso en el diagrama para que no sea tan pesado.
- Vease que para este diagrama solo se incluye un único trabajador (Map City Controller) y un único nodo de resumen de positivos por ciudad (Resume City Controller).

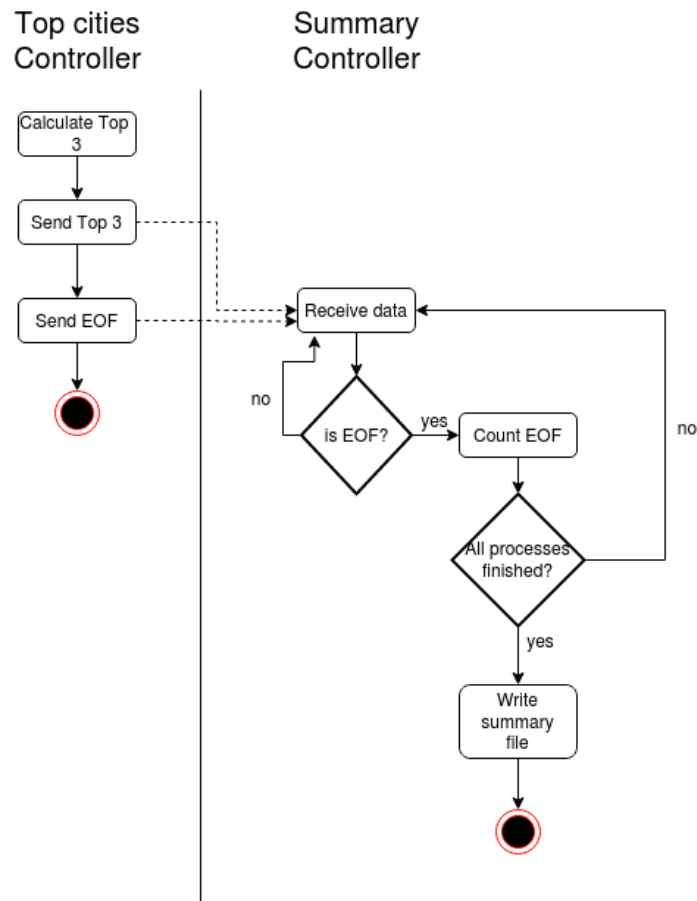


Figura 8: Diagrama de actividades del calculo de Top 3 ciudades. Parte 2/2

En esta segunda parte se ve el flujo de datos desde el calculo del top 3 de ciudades con más casos positivos hasta la escritura en el archivo. Se hace



una excepción asumiendo que es la única operación a realizar y por eso se realiza la escritura en el archivo.

## 7.2. Diagrama de secuencia

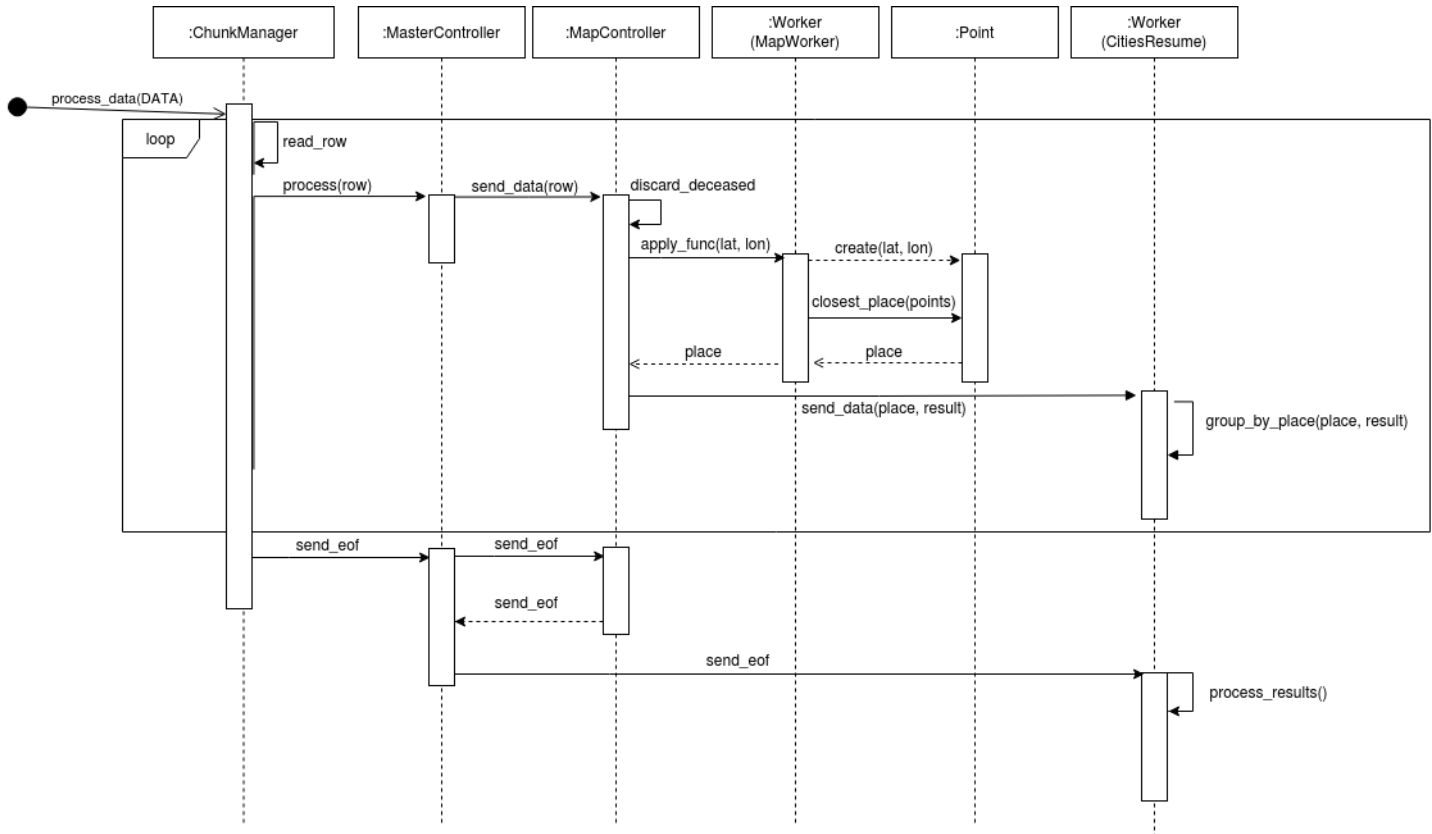


Figura 9: Diagrama de secuencia del calculo de Top 3 ciudades. Parte 1/2

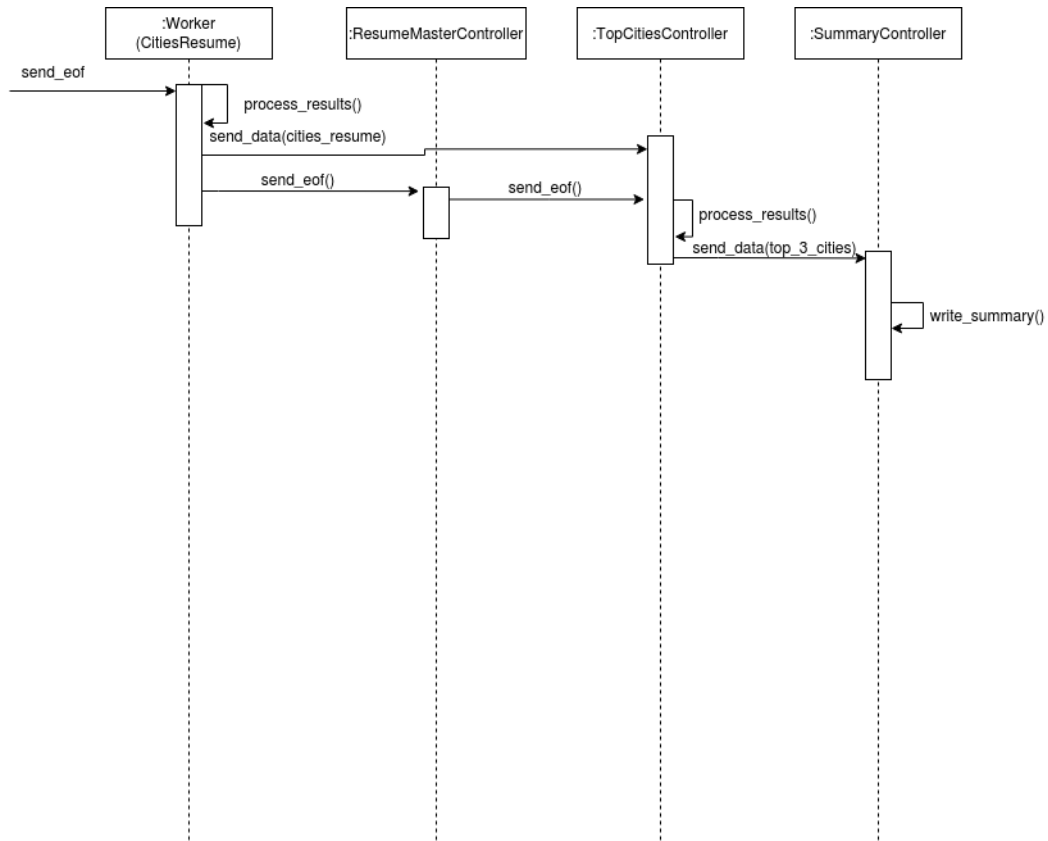


Figura 10: Diagrama de secuencia del calculo de Top 3 ciudades. Parte 2/2

En este diagrama se hace la misma suposición que en diagrama anterior, un único trabajador (Worker) y un único nodo de agrupación de datos (Resumer). Para que no sea un diagrama tan denso no se incluye el flujo de datos hacia el protocolo de comunicación, ya que este solo se encarga de recibir mensajes y transmitírselo a los próximos objetos.

## 8. Vista lógica

Se propone a mostrar los diferentes diagramas de clases que agrupan las clases de los paquetes mencionados en la sección de Vista de desarrollo. En muchos casos, estos diagramas no incluyen más de dos clases y son muy simi-

lares respecto a otros, en dichos casos no se mostrará el diagrama. Respecto al diagrama de estados, no se realizará uno ya que no hay muchos estados que compongan al sistema, en general puede verse el sistema como procesando información o leyendo un fin de archivo.

## 8.1. Diagrama de clases

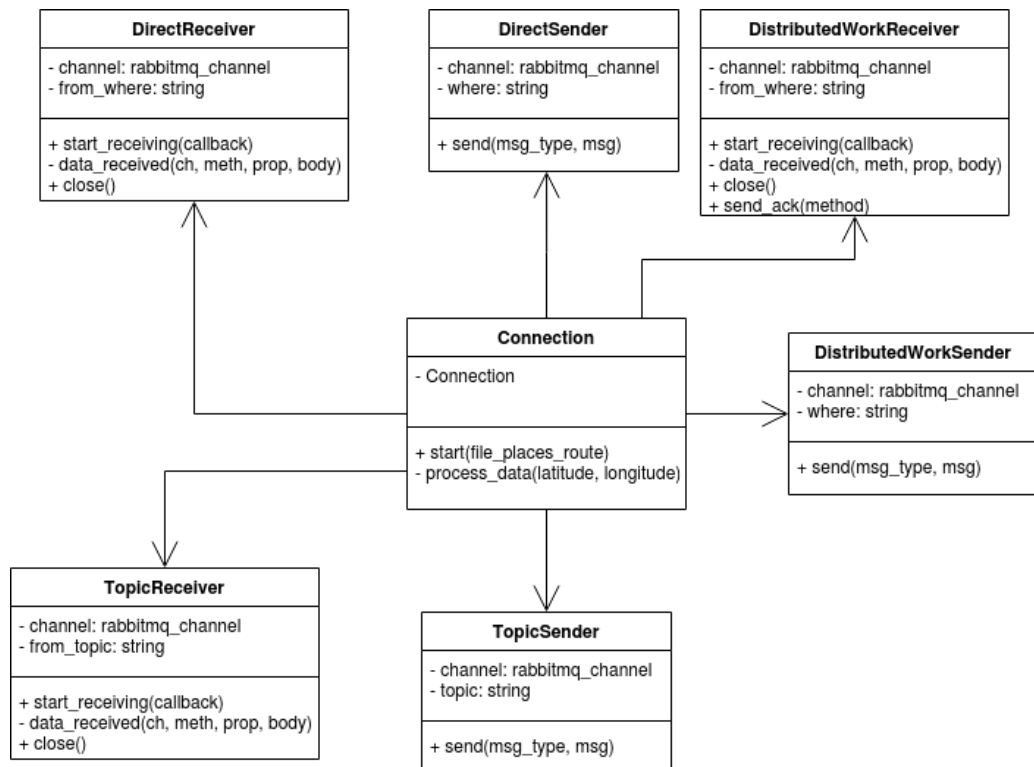


Figura 11: Diagrama de clases del Middleware

Descripción de clases:

- **Connection:** Maneja la conexión al middleware usando RabbitMQ. Se encarga de crear diferentes formas de comunicación entre procesos distribuidos.
- **DirectReceiver:** Crea un canal de comunicación para recibir información

de procesos (Comunicación punto a punto).

- **DirectSender**: Crea un canal para transmitir comunicación a un canal del tipo **DirectReceiver**.
- **DistributedWorkReceiver**: Crea un canal de comunicación para recibir información de varios que pueda ser leída por alguno de muchos procesos que utilicen este canal (Comunicación uno a uno con múltiples trabajadores).
- **DistributedWorkSender**: Crea un canal de comunicación para enviar información a un canal del tipo **DistributedWorkReceiver** para distribuir la carga de trabajo.
- **TopicReceiver**: Crea un canal de comunicación para recibir información bajo cierto tópico.
- **TopicSender**: Crea un canal de comunicación para enviar información bajo un cierto tópico.

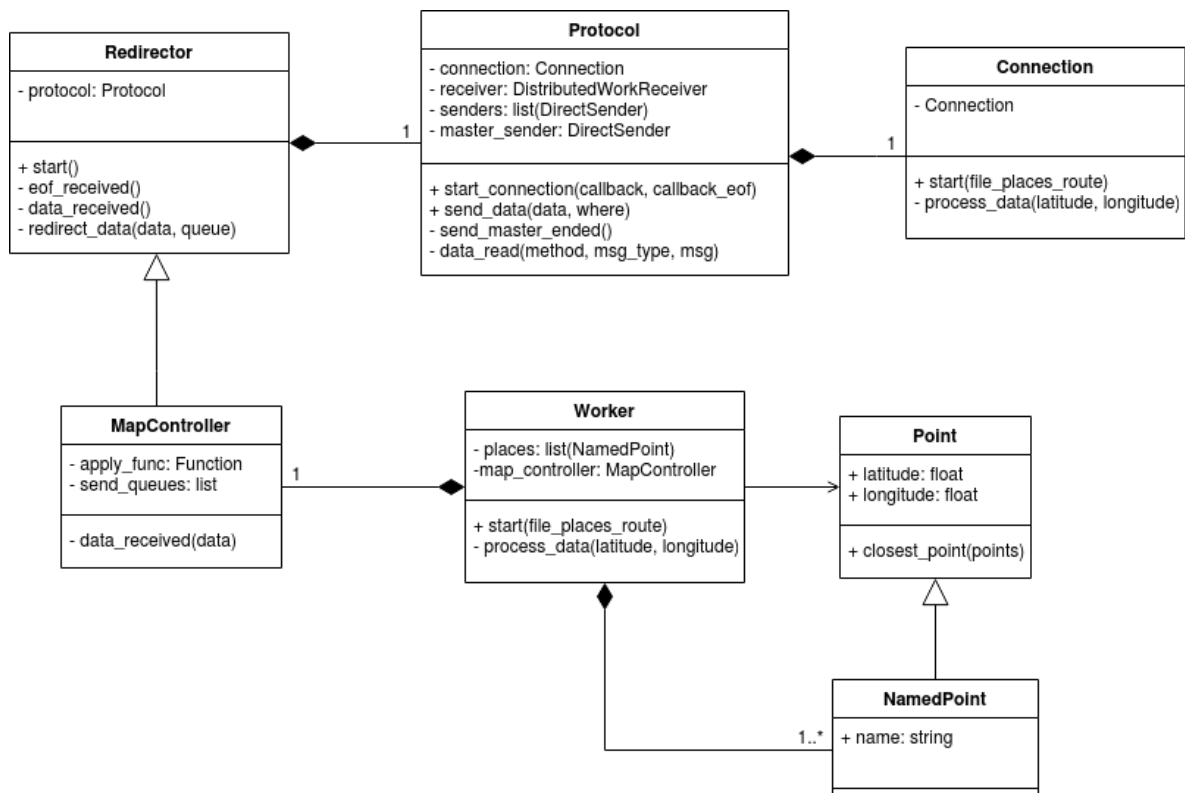


Figura 12: Diagrama de clases del Map Worker

Descripción de clases:

- **Worker:** Calcula la ciudad más cercana a la latitud y longitud recibida de un caso y procede a enviar los resultados a través del MapController.
- **Point:** Clase que representa un punto en el espacio, incluye un método para calcular el punto más cercano (Usando distancia de haversine) de una lista de puntos con respecto a si mismo.
- **NamedPoint:** Clase que representa un punto con nombre.
- **MapController:** Se encarga de recibir información a través del Redirector, aplicarle la función apply\_func y transmitir los resultados al próximo nodo que corresponda.

- Redirector: Clase abstracta que se encarga de recibir información a través de un canal de comunicación, enviar los resultados a alguno de múltiples canales y de enviar información de fin de archivo a un nodo maestro a través de un canal de comunicación especial.
- Protocol: Protocolo para manejar la creación y destrucción de canales de comunicación así como también manejar el envío de información, de fin de archivo y finalizar la comunicación.

Las clases del paquete del Middleware se sobreentienden que conectan con la clase Connection aquí mostrada, no se incluye para no hacer denso el gráfico. Las clases del paquete Redirector están incluidas en este diagrama (Redirector, Protocol y Connection), por lo que no se incluirá diagrama de clases para este paquete.

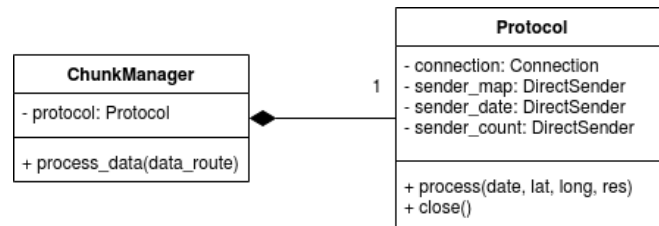


Figura 13: Diagrama de clases del Chunk Manager

Descripción de clases:

- ChunkManager: Se encarga de leer los datos de un archivo y transmitir cada fila a través del protocolo.
- Protocol: Maneja la creación de los canales de comunicación para la transmisión de datos a diversas colas para efectuar las operaciones.

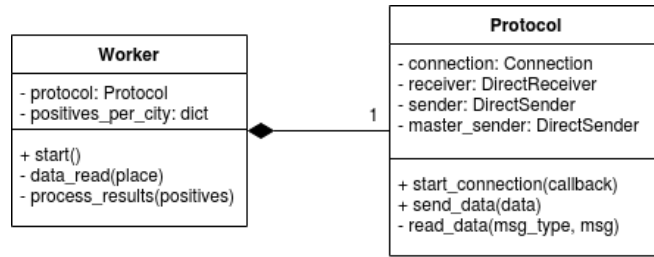


Figura 14: Diagrama de clases del Cities Resume

Descripción de clases:

- Worker: Se encarga de recibir la información de casos por ciudad y agruparlas por ciudad de manera de contar casos por ciudad, luego, al recibir toda la información procede a enviarla a la siguiente etapa.
- Protocol: Maneja la creación de los canales de comunicación para la transmisión de datos a diversas colas para efectuar las operaciones.

El resto de paquetes tienen todos un diagrama de clases similar, un worker que procesa la información, realiza alguna acción y envía la información a través de un Redirector (Que ya incluye un protocolo de comunicación) o a través de un propio Protocolo de comunicación. Por lo que no se incluyen diagramas de estos paquetes.

## 9. Manual de uso

Para utilizar este sistema, basta con ingresar el archivo de datos nombrado **data.csv** en la carpeta data dentro de **chunk\_manager**, los resultados serán escritos en el archivo: **summary\_controller/summary/summary.txt**

Para correr el programa, ejecutar:

*make docker-compose-up map\_workers=<TotalMapWorkers>*

*date\_workers=<TotalDateWorkers> count\_workers=<TotalCountWorkers>*

Indicando la cantidad de trabajadores que se utilizarán. Con respecto a la cantidad de nodos de agrupamiento (Date Summary Controllers y Cities Summary Controller) estos deben ser modificados manualmente, actualmente no está automatizada la forma de creación de los mismos y debe de insertarse en el código y en los archivos de docker-compose.

## 10. Problemas encontrados y mejoras disponibles

Este sistema presenta inconvenientes y mejoras disponibles a realizarse pero por falta de tiempo, no pudieron realizarse, entre ellas destacamos:

- Automatizar la creación de nodos de agrupación.
- Escribir clases más generales para no utilizar tantas clases de Protocolo (Refactor de código).
- Renombrar paquetes y clases a nombres más simples de entender (Refactor de código).
- No utilizar varios de canales de comunicación de salida del ChunkManager (Que funcionan de manera secuencial por el momento) y utilizar comunicación vía tópicos (Respecto a esto, hubo problemas por inicialización de colas para no perder información). De esta forma, se podrían agregar nuevas operaciones sin incluir un nuevo canal de comunicación a mano.
- Dado que para la prueba se utiliza una única computadora no muy potente, se incluye un único nodo de rabbitmq, sin embargo, este podría incluirse dentro de cada nodo de manera que no haya un único nodo



usando rabbitmq que pueda caerse y ser un punto de falla. Bastaría conectar las colas de entrada y salida utilizando direcciones para conectarse con los procesos de rabbitmq de cada nodo.

- Agrupar constantes fuera del código y permitir que el programa procese un archivo ingresado a través del comando y no ubicado en un determinado lugar y con un nombre en específico.
- Separar el archivo de docker-compose en múltiples archivos para cada operación, de manera que sea más legible y cómodo de usar.
- Sería útil incluir un nodo de monitoreo para monitorear el estado del sistema.
- Se incluye código para manejar configuraciones a través de archivo pero actualmente se están utilizando variables de entorno, sería mejor migrar a los archivos de configuración.