UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

Seguimento de contaminación de cidades españolas a partir de datos de TROPOMI

Autor/a:
Hugo Gómez Sabucedo

Titores:
Joaquín Ángel Triñanes Fernández

Grao en Enxeñaría Informática

Febreiro 2024

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría da Universidade de Santiago de Compostela para a obtención do Grao en Enxeñaría Informática

D. Joaquín Ángel Triñanes Fernández, Profesor/a do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela,

INFORMA:

Que a presente memoria, titulada *(Seguimento de contaminación de cidades españolas a partir de datos de TROPOMI)*, presentada por **D. Hugo Gómez Sabucedo** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa titoría no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a (Data):

Titor/a, Alumno/a,

(Joaquín Ángel Triñanes Fernández) (Hugo Gómez Sabucedo)

Agradecementos

Se se quere pór algún agradecemento, este vai aquí.

Resumo

Breve resumo das principais contribucións do traballo.

Índice xeral

Índice de figuras

Índice de cadros

Capítulo 1

Introdución

O cambio climático é un asunto que, actualmente, está á orde do día. O aumento das temperaturas a nivel global é un asunto que preocupa non só ós científicos, senón tamén á poboación en xeral. As causas do cambio climático son moitas, pero podemos estar de acordo en que unha gran parte é debida á presenza de múltiples gases ou substancias químicas que afectan á atmosfera.

Todos estes axentes afectan non só ó medio ambiente, senón ós seres humanos. Por iso, monitorizar a súa evolución é algo crucial, especialmente en entornos especialmente críticos, coma poden ser as cidades, nas cales a conxunción dunha gran densidade de poboación e movementos en áreas, polo xeral, pequenas, fai que sexa crucial coñecer cal é o nivel destes axentes contaminantes. Desta forma, as autoridades poden adoptar medidas, tanto no eido da saúde pública, para previr enfermidades asociadas a unha elevada presenza de ditos contaminantes, como de proteción do medio ambiente, para evitar que se causen danos ó entorno no que vivimos.

Así, dende a Axencia Espacial Europea (ESA) lanzouse, no ano 2017, un satélite de observación terreste nomeado Sentinel-5 Precursor (ou Sentinel-5P). Dito satélite, que se enmarca dentro do Programa Copernico, emprega un instrumento denominado espectrómetro, co obxectivo de monitorizar a contaminación atmosférica. Desta forma, TROPOMI proporciona observacións de forma diaria de gases atmosféricos que son chave á hora de monitorizar a calidade do aire e realizar predicións sobre a evolución da mesma, coma poden ser o metano (CH_4), monóxido de carbono (CO), formaldehído ($HCHO$), dióxido de nitróxeno (NO_2), ozono (O_3), aerosol ou dióxido de xofre (SO_2).

Este satélite proporciona datos a dous niveis distintos: L1, con datos crus; e L2, con variables xeofísicas derivadas dos datos de nivel 1. Existe outro nivel, o Nivel 3 (L3), no cal as variables están mapeadas en escalas regulares. Será este o nivel que nos interese alcanzar nos nosos datos, para poder obter con eles un agregado mensual que nos permita ver a media ou crear climatoloxías para poder observar anomalías nos mesmos.

O obxectivo deste traballo é, por tanto, empregar os datos que nos propor-

ciona este satélite para, mediante a realización de diferentes operacións sobre os mesmos, poder facer un seguimento da evolución ó longo do tempo da contaminación en diversas cidades de España. Decidiuse escoller este ámbito debido a que, ó tratarse dun área relativamente reducida, o procesamento dos datos será máis sinxelo, posto que será preciso procesar menos arquivos de menor tamaño.

1.1. Obxectivos do traballo

Os obxectivos deste traballo de fin de grao son os seguintes:

1. Mapear a contaminación atmosférica a partir de datos de sensores remotos, con especial relevancia en entornos urbanos

2. Monitorizar a variabilidade e tendencia dos diferentes parámetros satelitais relacionados coa contaminación do aire.

3. Desenvolver un visor online que permita representar de forma dinámica a información.

4. Implementar un sistema interoperable de distribución dos datos e produtos.

1.2. Estrutura da memoria

A presente memoria estrutúrase do seguinte xeito:

- **Capítulo 2:** Especificación de requisitos. Este capítulo describe detalladamente os requisitos e criterios que debe cumprir o proxecto.

- **Capítulo 3:** Deseño. Aquí explicaremos todo o relativo ó proceso de deseño do software ata acadar a súa versión final. Tamén se comenta a metodoloxía empregada para o seu desenvolvemento, así como as diferentes tecnoloxías das que se fixo uso no mesmo.

- **Capítulo 4:** Probas. Esta sección trata sobre as probas realizadas, así como os resultados das mesmas.

- **Capítulo 5:** Conclusións e posibles ampliacións. Neste último capítulo preséntanse as conclusións obtidas após realizar o proxecto, así como propostas de melloras sobre o mesmo ou posibles ampliacións futuras.

1.3. Descrición do sistema

Como se explicou nos obxectivos do traballo, o sistema que deseñemos terá como finalidade poder monitorizar a variabilidade e tendencia de diferentes parámetros, obtidos a través de datos de satélite, os cales se empregan para medir a contaminación do aire.

Será preciso, por tanto, deseñar un módulo mediante o cal poidamos descargar os datos de TROPOMI, obtidos a través da ESA, para poder traballar con eles, realizando as transformacións anteriormente mencionadas. Así, empregando as APIs proporcionadas, poderemos deseñar unha función que busque un determinado produto (por exemplo, NO_2) no catálogo de COPERNICUS, filtrando os resultados para centrarnos nun área de interese (neste caso, a Península Ibérica) e datas concretas. Construíndo unha query, e empregando a librería `requests`, descagaremos os arquivos para, posteriormente, transformalos. Ademais, os arquivos veñen en formato `.zip`, polo que será preciso, neste módulo, descomprimilos para poder obter os arquivos de formato `.nc` cos que debemos traballar. Este módulo será desenvolvido na linguaxe Python.

Por outra parte, deseñaremos outro módulo, tamén en Python, no cal poidamos traballar cos arquivos descargados. Os datos que descargamos son de nivel 2, polo que temos que realizar operacións cos mesmos para transformalos en datos agregados por mes. Para isto, será preciso empregar `HARP`, unha ferramenta que permite ler e procesar datos de satélite, permitindo a inxesta de arquivos e o traballo cos mesmos. Así, importaremos tódolos arquivos descargados e realizaremos as transformacións anteriormente citadas, para posteriomente exportar o resultado nun arquivo, tamén de formato `.nc`, nun directorio específico.

Os datos transformados almacenaranse nun servidor ERDDAP, que empregaremos como middleware. Este servidor proporciona unha forma sinxela de acceder a datasets científicos en formatos de arquivos comúns, facilitando a xeración de gráficos e mapas. Unifica diferentes tipos de servidores con diferentes estruturas e devolve os datos nun formato de arquivo común, así como estandariza as datas entre os mesmos. Ademais, é posible configurar un servidor ERDDAP propio e empregar os nosos propios datos. Esta será a opción que elixamos, xa que nos permite empregar os datos que nós mesmos transformamos.

Será preciso establecer, no servidor onde corra a aplicación, unha tarefa automática, para que, segundo se publiquen datos novos, estes estean dispoñibles na nosa aplicación. Esta tarefa deberá executarse tódolos meses, tendo en conta que os datos das medicións de satélite tardan uns días en estaren dispoñibles, e deberá descargar os datos dos parámetros que empregue a nosa aplicación, transformalos e almacenalos no directorio establecido.

Por último, deseñaremos unha interface web que permitirá consultar a evolución de cada un dos diferentes parámetros (escollendo un deles) ó longo do tempo.

Todo isto implementarémolo nunha máquina de AWS, de forma que o sistema

se administrará dende a mesma. Así, poderemos acceder á interface web de forma sinxela, a través de internet, e toda a xestión da aplicación estará tamén centralizada. Nesta máquina almacenaremos os datos transformados que obteñamos do módulo de Python, e nela teremos tamén o servidor ERDDAP co cal acceder a ditos datos. Accedendo á máquina poderemos visualizar a interface web que mencionamos.

Capítulo 2

Especificación de Requisitos

Neste capítulo trataremos en profundidade os principais requisitos que debe cumplir o sistema e o traballo; isto é, as funcionalidades que se require que teña o sistema. Especificaremos os usuarios que empregarán o sistema, as funcionalidades que se lle atribúen a cada un e os compoñentes do sistema, así como os requisitos funcionais e non funcionais do mesmo.

2.1. Identificación dos usuarios

O obxectivo deste proxecto é proporcionar información de diferentes parámetros relacionados coa contaminación atmosférica, co obexectivo de poder realizar un seguimento de como evolucionou ó longo do tempo a contaminación en diferentes puntos da Península Ibérica. Temos, por tanto, un perfil de usuario claro: aquel que consultará os datos, ou usuario final [U1]. Por outra parte, identificaremos outro usuario, administrador [U2] que será o encargado de realizar o mantemento de todo o sistema, e que se encargará de arranxar os distintos problemas que poidan surxir co uso do mesmo.

- O **usuario final ([U1])** é aquela persona á que vai dirixida principalmente o sistema, e que ten interese en poder monitorizar a evolución da contaminación por diferentes parámetros ó longo do tempo. Trátase dun perfil de persoa variada, dende un científico, un traballador público que queira elaborar plans para a redución da contaminación nas cidades, ou calquera cidadán con interese polo tema. Porén, as tarefas que se lle atribúen son sinxelas:

    1. **Consultar o nivel de contaminación.** Este usuario consultará, na interface web do servidor en que se desenvolva o sistema, o valor dun parámetro en concreto, de entre tódolos que estean dispoñibles, para un mes concreto.

- O **administrador do sistema** [**U2**] é a persoa que se encargará do mantemento do mesmo, monitorizando o funcionamento do servidor para que, en caso de que se produza algún erro, poder solventalo. O rol de administrador do sistema asumiráo o alumno encargado de desenvolver este traballo. As súas principais tarefas serán:

  1. **Configurar o sistema**. Este usuario será o encargado de realizar a configuración inicial do sistema, implementando os diferentes módulos que o compoñen e descargando e procesando datos durante un periodo de tempo o suficientemente longo, que permita a correcta utilización do sistema.
  2. **Monitorear o sistema**. Como administrador, deberá encargarse de monitorizar o correcto funcionamento do sistema, vixiando os posibles problemas que poidan xurdir no mesmo. Especialmente, deberá facer énfase en controlar que, cada mes, se descarguen e procesen correctamente os arquivos correspondentes, para asegurar a máxima calidade nos datos que consulte o usuario final.

2.2. Requisitos funcionais

2.2.1. Obter datos de satélite da ESA [RF1]

A aplicación que se desenvolva debe ser capaz de obter, de forma automática, os datos de satélite dispoñibles no dataspace de COPERNICUS, na web da Axencia Espacial Europea. Isto farase empregando unha API de entre aquelas que ofrece a propia ESA. Estes datos deberán limitarse a unha área xeográfica de interese, que no noso caso será a Península Ibérica, e a un rango de datas determinados.

2.2.2. Transformar e agregar datos de satélite [RF2]

Os datos que se descarguen dende a web da ESA deberán ser transformados, de forma que serán agregados mensualmente para poder facer un seguemento da evolución da contaminación. Os arquivos resultantes deberán manter unicamente a información necesaria.

2.2.3. Implementar un servidor ERDDAP local [RF3]

No servidor onde se implemente este sistema, implementarase un servidor ERDDAP local, de forma que se facilite o acceso ós datos. Deberase configurar un *dataset* por cada un dos parámetros, e almacenar nel os diferentes arquivos.

2.3. Requisitos do produto

2.3.1. A aplicación deberá ser rápida e eficiente [RNF1]

A aplicación deberá ser capaz de, nun tempo razoable dende que estean dispoñibles, descargar e procesar os datos de satélite, de forma que os usuarios poidan acceder a estos datos o antes posible.

2.3.2. Eficiencia de almacenamento [RNF2]

A aplicación debe ser eficiente en canto ó uso do almacenamento se refire. Tendo en conta que, aproximadamente, cada mes se descagarán en torno a 50GB de novos datos por cada produto que se procese, e que após o procesamento teremos datos cun tamaño resultante darredor de 500MB por parámetro, é crucial que a aplicación almacene só os datos estritamente necesarios.

2.4. Requisitos de rendemento

2.4.1. Tempo de resposta razoable [RNF3]

A aplicación debe ter un tempo de resposta razoable, de forma que os usuarios non deban agardar moito tempo para teren os resultados que desexan.

Capítulo 3

Deseño

Neste capítulo, explicaremos os diferentes compoñentes que forman o sistema e as tecnoloxías empregadas, así como o proceso de desenvolvemento e implementación do mesmo.

3.1. Compoñentes do sistema

No que respecta á arquitectura do sistema software desenvolvido, podemos identificar tres grandes compoñentes:

- **Backend**. O backend da aplicación está formado polos diferentes scripts de Python que se encargan da descargar os datos, mediante a API, da web de COPERNICUS, da súa descompresión, o procesamento e o almacenamento no directorio apropiado para que se poida acceder a eles. Ademais, cun script de Python e o programa `cron` de Ubuntu, definiremos unha tarefa a executarse automáticamente de forma mensual, de forma que, segundo teñamos dispoñibles novos datos, estes se descarguen e sexan accesibles ó usuario, para que teña sempre os datos máis recentes.

- **Middleware**. O middleware está formado polo servidor ERDDAP, que serve como unha capa intermedia entre o backend de Python e o frontend que verá o usuario final. Ademais, como comentamos na introdución, proporciona unha forma sinxela de acceder a datasets científicos en formatos de arquivos comúns, facilitando a xeración de gráficos e mapas. Coa implementación do noso propio servidor ERDDAP, poderemos almacenar nel, ademais dos datasets que xa veñen por defecto, uns cos nosos arquivos.

- **Frontend**. O frontend desenvolverase empregando HTML. Será un frontend sinxelo, de forma que, cando se acceda á aplicación na URL do servidor, o usuario vexa un mapa, acoutado á zona na que se realiza o estudo dos datos, e que poida elixir, por unha parte, un parámetro de entre os catro

que se proporcionan, e un mes no rango de meses dispoñibles no dataset, actualizándose a información de forma dinámica.

3.1.1. Backend

Como xa comentamos, o backend da aplicación desenvolveuse en Python. A elección desta linguaxe de programación para este módulo é sinxela: é a linguaxe de programación máis axeitada para traballar con grandes cantidades de datos. Ademais, é a linguaxe que ten maior compatibilidade con tódalas ferramentas que deberemos empregar para o procesado dos datos, o que fai que sexa, practicamente, a única opción a ter en conta..

Así, o backend de Python divídese en catro scripts diferentes, cada un cunha funcionalidade determinada.

Script principal e automatización

Por unha parte, temos o script principal, `app.py`. Este é o script dende o que se executa o programa. Nel, establécense os parámetros cos cales se executaran os módulos de descarga e procesado dos arquivos, comprobando tamén a súa validez (como, por exemplo, que a data de inicio da busca sexa anterior á data de fin). Pódese executar tanto por liña de comandos coma importando o arquivo e chamando á función.

Por outra parte, temos o script `auto.py`, o cal se encarga da automatización da descarga e procesado dos datos. Este será o script que executaremos con `cron` de forma mensual, para manter os datasets actualizados. Este script obtén, a partir da data actual, as datas para as que realizaremos a busca, que serán entre o primeiro e último día do mes anterior. Desta forma, executaremos o *job* de `cron`, cunha frecuencia mensual, o día 10 de cada mes. Polo tanto, no caso da execución que se realizaría o día 10 de xaneiro de 2024, as datas que se tomarían como referencia serían o 1 de decembro de 2023 como data de inicio, e o 31 de decembro de 2023 como data de fin. Decidiuse elixir o décimo día de cada mes para a actualización dos datos debido a que, como se explica en [1], o procesado dos arquivos resultantes das medicións dos satélites ten unha latencia de 5 días. Así, asegurámonos de que, cando se execute o script, tódolos arquivos que se deben ter en conta estarán dispoñibles.

Script de descarga de datos

En segundo lugar, temos o script de descarga de datos. A súa función principal é `obtenArquivos`, que recibe como parámetros a area de interese da busca, as datas de inicio e de fin, o parámetro para o que se realizará a busca e o directorio onde se almacenarán os arquivos. Se este directorio non existe, creao.

A continuación, empregando a API de oData, realiza unha consulta para obter os arquivos que se descargarán. Iterando por estes arquivos, obten o seu Id e o nome e, se estos non existen (tanto en formato zip, por estaren comprimidos, ou en formato zip, por estaren descomprimidos), realiza a súa descarga, na función `descargaArquivo`. Posteriormente, descomprime o arquivo coa función `descomprimeArquivo`, os cales quedan dentro dunha carpeta co seu nome. Por tanto, unha vez descargados e descomprimidos tódolos arquivos, móveos á carpeta principal e elimina tódalas subcarpetas.

No que respecta á función de descarga de arquivos, en primeiro lugar chama a unha función para obter o token de autorización preciso para descargar os datos. A continuación, crea a url para a descarga e, mediante a libraría Session, descarga o arquivo, sempre que a resposta do servidor sexa positiva. Por outra parte, a función de descompresión dos arquivos comproba, en primeiro lugar, que o ficheiro zip exista. Se é así, empregando a libraría zipfile, extrae os arquivos no directorio (imprimindo unha mensaxe de erro en caso de que o arquivo sexa inválido) e, unha vez feito, bórrao.

Script de procesado de datos

Por último, temos o script que se encarga de procesar os datos. Nel, o módulo principal que empregamos é HARP que, como se explica en 3.2.2, é unha ferramenta que permite a lectura e procesado de datos de satélite. Neste módulo temos diferentes funcións de procesado, unha para cada un dos diferentes parámetros que se ofrecen na aplicación, xa que as operacións deben personalizarse en función do parámetro. Así, establécese unha función xenérica que, en función do parámetro que se estea tratando, invoca a unha función ou outra e, unha vez procesados tódolos arquivos e exportados os datos, elimina os arquivos que se descargaron, co obxectivo de optimizar o almacenamento do servidor (posto que estamos falando de centenares de arquivos cada mes, con tamaños de entre 500MB e 1GB, os cales, unha vez procesados, non son de interese nin utilidade). Ademais, temos unha función denominada `obtenListaProdutos` , para obter os arquivos a procesar.

Malia que cada unha das funcións de procesado son bastante semellantes, posto que realizan unha serie de operación similares entre elas, foi preciso particularizalas, xa que o nome das variables que exportaremos difire en función do parámetro. Estas funcións reciben unha lista de arquivos a procesar e un directorio onde almacenar o arquivo final procesado. Empregando a función `import_product` de harp, importamos os arquivos, realizando con eles unha serie de operacións para filtrar os datos segundo a súa calidade, como se indica nos diferentes manuais. Unha vez temos tódolos produtos importados, realizamos unha serie de operacións, propias para cada produto, para quedarnos cos datos dunha zona e obter con eles unha cuadrícula; e unha serie de post-operacións, globais a tódalas funcións, para realizar a agregación dos datos. A continuación, definimos o nome do arquivo a exportar (que segue o patrón parametro_AAAAMM.nc), e gardámolo.

3.1.2. Middleware

O middleware que empregamos será ERDDAP. Isto é un servidor de datos que proporciona unha forma simple e consistente de acceder e descargar múltiples datasets científicos, de diferentes fontes, nun formato común, facilitando a elaboración de gráficos e mapas [2]. Ademais, permite obter os datos en diversos formatos de arquivo, ademais de proporcionar funcións para estandarizar as datas dos resultados ou proporcionar os mesmos en formatos de imaxe personalizados. A instalación de ERDDAP de referencia é aquela feita pola National Oceanic and Atmospheric Administration (NOAA, Oficina Nacional de Administración Oceánica e Atmosférica) que, ademais de ter máis de 200 datasets, permite crear un servidor local para almacenar nel datos propios.

Para realizar esta instalación, só será necesario seguir os pasos que se indican en [3], o que explicaremos máis detalladamente en 3.3 e 3.4. En resumo, é preciso:

1. Dispoñer dunha versión de Java 17.

2. Configurar Tomcat.

3. Descargar os arquivos de configuración de ERDDAP e realizar os cambios precisos para adaptalo ó sistema.

4. Instalar o arquivo `.war` de ERDDAP.

5. Iniciar o servidor e configurar os datasets que queremos que sexan visibles no mesmo.

3.1.3. Frontend

Para o frontend do proxecto, crearemos unha aplicación sinxela en Tomcat, empregando HTML. Facendo uso da funcionalidade que nos proporciona ERDDAP para xerar imaxes a partir dos datos, crearemos unha interface na que teñamos, por unha parte, un mapa cos datos que queiramos visualizar e, por outra parte, dous selectores, un para seleccioanr o parámetro a consultar e outro para seleccionar o mes e ano.

3.2. Tecnoloxías empregadas

3.2.1. Anaconda

3.2.2. HARP

3.3. Desenvolvemento do sistema

3.3.1. Introdución

Para poder comezar a desenvolver o sistema, primeiro foi preciso realizar unha introdución á detección mediante satélite de diferentes datos e ó funcionamento dos satélites de TROPOMI. Para isto, foron de utilidade os seminario dispoñibles en [5], de monitorización de alta resolución de NO_2 dende o espacio con TROPOMI; [6], de ferramentas para analizar datos de alta calidade de satélite; e [7], de aplicacións de observacións de satélites para a calidade do aire e a exposición á saúde. Neles, explícase o funcionamento básico dos satélites e como miden e procesan os datos. Tamén se explican os diferentes niveis de datos que existen, como mencionamos na introdución. Dipoñemos de 4 niveis principais de datos:

1. **Nivel 0 (L0):** estes datos son datos crus, tal e como se obteñen das medicións do satélite, a máxima resolución.

2. **Nivel 1 (L1):** son os datos de nivel L0, pero con referencias temporais e información dos sensores empregados para as medicións, coma coeficientes de calibración radiometrica e xeometrica.

3. **Nivel 2 (L2):** os datos de nivel L1, pero con variables xeofísicas derivadas na mesma resolución e ubicación que os datos de nivel L1.

4. **Nivel 3 (L3):** as variables obtidas dos datos de nivel L2, mapeadas en forma de cuadrículas espazo-temporais, completas e con consistencia.

Empregar datos de nivel L3 ten diversas vantaxes:

- Podemos dispoñer dun único arquivo por día (ou por mes, ou o período de tempo que nos interese). Polas características do satélite, o seu ciclo orbital é de 16 días, realizando 14 órbitas por día [9]. Polo tanto, para dous días distintos, é posible que un arquivo non cubra a totalidade da área de interese. Ó realizar as operacións de agregación dos datos, aseguramos ter un único arquivo por día, facendo a manipulación dos datos moito máis sinxela.

- Datos dispostos en cuadrículas uniformes. Coa agregación dos datos, podemos crear tamén unha cuadrícula para os mesmos, dispoñéndoos de forma regular no mapa ou área que nos interese estudar.

- Arquivos de menor tamaño. Os arquivos de nivel L2 conteñen unha maior cantidade de datos, unha gran parte dos cales pode non ser relevante para o noso sistema. O seu procesamento a nivel L3 permite elimar estos datos e variables innecesarias, minguando o tamaño dos arquivos.
- Maior calidade nos datos. Nos diferentes manuais dos produtos, establécense diversos criterios de calidade asociados ós datos, que podemos empregar para descartar medicións de certas áreas pequenas do espazo en caso de que non teñan unha calidade suficiente. Desta forma, ó procesar os datos, podemos eliminar estas medicións de menor calidade e, como crearemos agregados por grandes períodos de tempo, evitaremos que queden vacíos.

O satélite SENTINEL 5P proporciona datos de seis parámetros moi relacionados coa contaminación atmosférica: ozono (O_3), tanto a columna troposférica (aquela que se forma en capas mais altas da atmosfera) coma a columna total; dióxido de nitróxeno (NO_2), tanto a columna troposférica coma a total; dióxido de xofre (SO_2), a columna total; monóxido de carbono (CO), a columna total; metano (CH_4), a columna total; e formaldehido ($HCHO$), a columna total. Tratar todos estes parámetros era difícil de abordar, polo que foi preciso descartar dous deles para non telos en conta. Baseándonos na información dispoñible en [10], decidimos elixir os seguintes: NO_2, CO, SO_2 e O_3. Escolléronse estes parámetros por seren os que máis problemas de saúde adoitan causar, ademais de por empregárense máis a miúdo nos diferentes mapas de contaminación.

3.3.2. Descarga de datos

Unha vez decididos os parámetros, podemos comezar coa descarga dos datos. Para isto, no Dataspace Copernicus temos dispoñibles múltiples APIs que podemos empregar para acceder e descargar os datos . Estas APIs poden ser de tres tipos: *Catalogue APIs*, que permiten buscar no catálogo de produtos dispoñibles e descargar os arquivos; *Streamlined Data Access (EDA) APIs*, que permiten obter datos de observación terrestre (EO) xunto con ferramentas de procesamento e análise; e outras APIs, que permiten, por exemplo, o acceso a datos EO empregando *buckets* de S3. No noso caso, elixiremos unha API do primeiro tipo, xa que é preciso poder buscar e filtrar os produtos para descargar aqueles que nos interesen.

Empregaremos a API de OData, que está baseada en APIs REST e permite, empregando mensaxes HTTP sinxelas, publicar e editar recursos identificados con URLs. A documentación completa da API atópase en [11]. En xeral, unha query OData contén a URL base xunto cunha serie de opcións, entre as que se atopan: *filter*, *orderBy*, *top*, *skip*, *count* e *expand*. Dentro da opción de *filter*, podemos filtrar por:

1. **Nome do produto:** filter=Name. Buscando un produto específico polo seu nome.

2. **Colección de produtos:** filter=Collection/Name. Buscar produtos dunha colección concreta (SENTINEL-1, SENTINEL-2, SENTINEL-3, SENTINEL-5P, SENTINEL-6 ou SENTINEL-1-RTC).

3. **Data de publicación:** filter=PublicationDate. Filtrando entre un intervalo de datas, que pode incluír (con `ge` e `le`) ou non (con `gt` e `lt`) as datas que se empreguen.

4. **Data de detección:** filter=ContentDate/Start / filter=ContentDate/End. Filtrando pola data de detección dos datos.

5. **Área xeográfica:** filter=OData.CSC.Intersects(area=geography'SRID=4326;). Pódense filtrar os datos para elixir aqueles que crucen cun punto (`POINT(x,y)`) ou con polígono (`POLYGON()`, cunha lista de puntos).

Por outra parte, existen as seguintes opcións, que deben ir separadas dos filtros e entre elas empregando o símbolo $:

- **orderBy**: para ordenar de forma descendente (`desc`) ou ascendente (`asc`), que é a opción por defecto. Pódese filtrar por *ContentDate/Start*, *ContentDate/End*, *PublicationDate* ou *ModificationDate*.

- **top**: para limitar o número máximo de ítems que devolverá a query. Por defecto é 20, e pode establecerse como un número enteiro entre 0 e 1000.

- **skip**: permite saltar un certo número de resultados para, por exemplo, poder paxinar en caso de que a query obteña moitos resultados. O valor por defecto é 0, e pode tomar un valor entre 0 e 1000.

- **count**: para obter o número exacto de produtos que coincidan coa query. Por defecto, está establecida a `False`, deshabilitada.

- **expand**: permite ver os metadatos completos de cada un dos resultados que obtivo a query. Acepta como argumentos *Attributes* ou *Assests*.

Para realizar a busca dos produtos, unicamente é preciso empregar unha libraría que nos permita facer peticións HTTP. Neste caso, elixiuse a libraría `requests` de Python, xa que permite realizar estas peticións dunha forma moi sinxela. A URL base dende a que obteremos os datos é `https://catalogue.dataspace.copernicus.eu/odata/v1/Products`, engadiremos os seguintes filtros:

- **Área xeográfica:** limitaremos a busca á Península Ibérica, para o que definiremos o seguinte polígono: `POLYGON((-9.647785 35.912135,-9.647785 44.230843,4.916224 44.230843, 4.916224 35.912135,-9.647785 35.912135))`. Esta área é a que se mostra na imaxe 3.1

- **Colección de produtos:** filtramos os produtos da colección `SENTINEL-5P`.

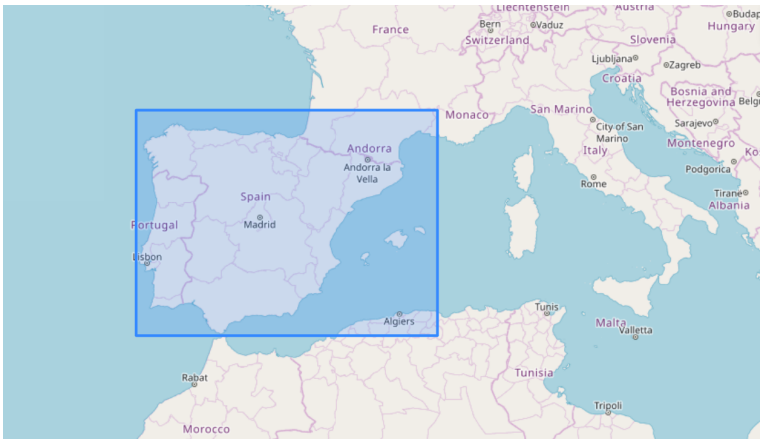

Figura 3.1: Area de interese sobre a que se realiza a busca.

- **Nome de produto:** empregamos o filtro `contains(Name, 'S5P_OFFL_{parametro}')`, para obter os produtos do parámetro que nos interese.
- **Data de detección:** coas datas que se lle pasan á función dende o módulo principal da aplicación, filtramos en base á data de detección dos datos xa que, como comentamos en 3.1.1, os datos tardan varios días en publicarse dende que se miden.
- Aplicamos ademais a opción `top` para obter un número maior de resultados, que estableceremos a 100. Isto débese a que temos, de media, dous produtos por cada día que se busca e, no noso caso, ó facer buscas mensuais, o número de arquivos que se descargan será en torno a 60, moi superior ó límite de 20 que establece por defecto a API.

Os resultados obtidos desta petición almacénanse nunha variable que se convertirá a un DataFrame de `pandas`, que contén, entre outros, o nome do arquivo, o seu identificador, a data de detección e publicación ou a pegada xeográfica. Así, iremos iterando sobre o DataFrame para, empregando o Id do produto e o seu nome, descargar os diferentes arquivos.

Para poder descargar os produtos, é preciso xerar un token de autenticación, xa que é preciso estar rexistrado. O proceso de rexistro é moi sinxelo, e pode facerse dende a páxina principal de COPERNICUS. No noso caso, decidiuse elixir o correo corporativo da universidade, xunto cun contrasinal xenérico. Foi preciso tamén crear unha función, `get_keycloak`, que, recibindo como parámetros o usuario e o contrasinal, devolve o token de acceso. É preciso ter en conta as limitacións e cuotas que se explican en [12], onde se explica que cada token de acceso é válido durante 10 minutos, podendo ter un máximo de 100 sesións activas simultaneamente. Por iso, foi preciso xerar o token de acceso na propia función de descarga, e non antes de realizar a mesma xa que, desta forma, corríase o risco de

que a sesión expirase antes de poder descargar tódolos arquivos, o cal provocaría un fallo.

Unha vez temos o token de autenticación, creamos un obxecto de tipo `Session`, no cal estableceremos, mediante o método `.headers.update()`, o token de autenticación. A continuación, establecemos a url na que se atopa o arquivo, que será `https://catalogue.dataspace.copernicus.eu/odata/v1/Products(id)/$value`, onde id é o identificador do produto que vamos descargar. Antes de descargar o arquivo, debemos comprobar o código de estado de HTTP da resposta, para comprobar que todo sexa correcto. Desta forma, descargaremos o arquivo se o código de estado é 200 (OK) ou 308 (Permanent redirect). Se o estado é dos tipos 30X, significa que precisamos accións adicionais para completar a petición, polo que volveremos a repetila. Se o estado é correcto, escribiremos o contido do arquivo nun ficheiro .zip, no directorio que corresponda.

Inmediatamente despois de descargar o arquivo, temos que descomprimilo posto que, como vimos de comentar, é preciso descargalo en formato ZIP. Para isto, temos unha función moi sinxela, denominada `descomprimeArquivo`, que comproba que o arquivo exista e emprega a libraría zipfile para extraelo, comprobando tamén que sexa un arquivo zip válido (capturando a excepción *zipfile .BadZipFile*). Unha vez descomprimido, eliminaremos o ficheiro zip, posto que xa non será necesario. Debemos ter en conta que o ficheiro final que queremos obter, en formato netCDF (ou .mc) non se atopará directamente no directorio en que se extrae, senón que se atopa dentro dunha subcarpeta co nome do arquivo que acabamos de descargar. Por tanto, unha vez descargamos e descomprimimos tódolos arquivos, deberemos mover os ficheiros .nc ó directorio de descarga, podendo eliminar posteriormente tódolos subdirectorios.

3.3.3. Procesamento de datos

Como xa comentamos, para procesar os datos empregaremos, principalmente, HARP 3.2.2, unha ferramenta de lectura e procesado de datos de satélite, que se pode empregar en Python de forma moi sinxela. A idea básica é importar unha serie de produtos, ós cales se lles aplican unha serie de operacións e postoperacións para transformalos en arquivos en cuadrículas regulares, e posteriormente exportalos. Así, deseñamos unha función xenérica `transformaL3`, que recibe a ruta onde están os arquivos a procesar e o tipo de produto dos mesmos, e a ruta onde se exportar á o arquivo resultante. Así, podemos optimizar o código, realizando dende esta función a chamada a cada unha das funcións deseñadas para cada un dos parámetros. Unha vez se procesa o arquivo en cuestión, eliminaranse tódolos arquivos que se empregaron nesta operación, para cumplir co requisito [RNF2].

A continuación, explicarase máis detalladamente o proceso de transformación de cada un dos produtos.

Transformación dos datos de *CO*

A función de procesado dos datos de monóxido de carbono recibe, ó igual que as demais, unha lista de arquivos e unha ruta onde almacenar os arquivos procesados. Con esta lista de arquivos, podemos iterar sobre ela, de forma que importamos os arquivos de un en un. Isto faise así, e non todos de golpe, para que o proceso de importado sexa máis eficiente e previr fallos ó ter que importar numerosos arquivos simultáneamente. Así, empregamos a función `import_products` de HARP, a cal nos devolve un produto de HARP que engadiremos a unha lista. Esta función pode recibir unha serie de operacións que se aplicarán ós arquivos á hora de importalos. A lista das posibles operacións que se poden empregar en HARP atópase en [13]. No caso do monóxido de carbono, temos as seguintes operacións:

1. `CO_column_number_density_validity` 50 : con isto, filtramos os datos en base á calidade dos mesmos. Segundo as indicacións do manual de producto de CO ([18]), recoméndase ignorar os datos cun valor qa_value menor que 0.5. Porén, como se explica en [14], a variable qa_value é mapeada como CO_column_number_density_validity.

2. `derive(CO_column_number_density [Pmolec/cm2]`: a operación derive serve para cambiar o tipo de datos ou unidade dunha variable. Neste caso, cambiaremos o valor da columna de monóxido de carbono (que no arquivo orixinal se denomina carbonmonoxide_total_column) a petamoléculas por centímetro cadrado, xa que esta vén en moles por metro cadrado, para traballar con valores numéricos máis pequenos.

3. `keep(latitude, longitude, latitude_bounds, longitude_bounds, CO_column_number_density)`: con esta operación, indicamos que queremos incluír estas variables no produto.

Unha vez importamos tódolos produtos, empregaremos a función `execute_operations` para realizar operacións sobre os mesmos. Ó pasar unha lista de produtos á función, estes agruparanse despois de realizar as operacións sobre os mesmos. Neste caso, as operacións a realizar son:

1. `bin_spatial(2001, 35, 0.005, 3001, -10, 0.005)`: esta operación mapea tódalas observacións nunha cuadrícula espacial de latitude/lonxitude. Neste caso, creamos unha cuadrícula de 2001 puntos de alto, comezando no paralelo 35°, cun *offset* de 0.005°; e 3001 puntos de ancho, comezando no meridiano -10°, cun *offset* de 0.005.

2. `exclude(latitude_bounds_weight, longitude_bounds_weight, weight, latitude_weight, longitude_weight)`: excluímos estas variables do produto final.

3. `derive(latitude{latitude})` e `derive(longitude{longitude})`: derivamos as coordenadas de latitude e lonxitude da nova cuadrícula.

Por último, aplicamos unha serie de post-operacións, que definimos de forma global nunha variable, xa que serán as mesmas para tódolos produtos:

1. `bin()`: realiza un promedio de tódalas variables do produto na dimensión temporal.

2. `squash(time, (latitude, longitude))`: elimina a dimensión *time* para unha lista de variables (neste caso, *latitude* e *longitude*).

Unha vez temos o arquivo resultante, almacenámolo no directorio indicado. Para obter o nome do arquivo, seguiremos a expresión `CO_AAAAMM`, onde AAAA é o ano e MM é o mes do arquivo que exportamos. Isto obtémolo directamente do nome de calquera un dos arquivos que empregamos.

Transformación dos datos de NO_2

Neste caso, o proceso é igual ó anterior. Os nomes das variables mapeadas atópanse dispoñibles en [15]. As operacións que aplicamos ó importar os produtos son:

1. `tropospheric_NO2_column_number_density_validity 75`: neste caso, o manual de dióxido de nitróxeno ([19]) indícanos que debemos considerar os datos cun valor de qa_value maior a 0.75.

2. `derive(tropospheric_NO2_column_number_density [Pmolec/cm2])`: derivamos a variable que orixinalmente se denomina nitrogendioxide_tropospheric_column das súas unidades orixinais (mol/m^2) a $Pmolec/cm^2$.

3. `keep(latitude, longitude, latitude_bounds, longitude_bounds, tropospheric_NO2_column_number_density)`: incluímos as variables no produto.

As operacións que se executan sobre os datos son:

1. `bin_spatial(2001, 35, 0.005, 3001, -10, 0.005)`: creamos unha cuadrícula de 2001 puntos de alto, comezando no paralelo 35°, cun *offset* de 0.005°;

2. e 3001 puntos de ancho, comezando no meridiano -10°, cun *offset* de 0.005.

3. `exclude(latitude_bounds_weight, longitude_bounds_weight, weight, latitude_weight, longitude_weight)`: excluímos estas variables do produto final.

4. `derive(latitude{latitude})` e `derive(longitude{longitude})`: derivamos as coordenadas de latitude e lonxitude da nova cuadrícula.

Aplicamos como post-operacións as mesmas que no anterior caso. O arquivo resultante almacénase no directorio indicado, e o seu nome segue a expresión `NO2_AAAAMM`, onde AAAA é o ano e MM é o mes o arquivo que exportamos.

Transformación dos datos de O_3

Para os datos do ozono, aplicamos as seguintes operacións ó importar os produtos:

1. `O3_column_number_density_validity 50`: neste caso, o manual do ozono ([20]) indícanos que debemos considerar os datos cun valor de qa_value maior que 0.5.

2. `derive(O3_column_number_density [Pmolec/cm2])`: derivamos a variable que orixinalmente se denomina ozone_total_vertical_column das súas unidades orixinais (mol/m^2) a $Pmolec/cm^2$.

3. `keep(latitude, longitude, latitude_bounds, longitude_bounds, O3_column_number_density)`: incluímos as variables no produto.

Unha vez importados, executamos as seguintes operacións:

1. `bin_spatial(2001, 35, 0.005, 3001, -10, 0.005)`: creamos unha cuadrícula de 2001 puntos de alto, comezando no paralelo 35°, cun *offset* de 0.005°;

2. e 3001 puntos de ancho, comezando no meridiano -10°, cun *offset* de 0.005.

3. `exclude(latitude_bounds_weight, longitude_bounds_weight, weight, latitude_weight, longitude_weight)`: excluímos estas variables do produto final.

4. `derive(latitude{latitude})` e `derive(longitude{longitude})`: derivamos as coordenadas de latitude e lonxitude da nova cuadrícula.

As post-operacións que se aplican son as mesmas que nos anteriores casos. O arquivo resultante almacénase no directorio indicado, e o seu nome segue a expresión `O3_AAAAMM`, onde AAAA é o ano e MM é o mes o arquivo que exportamos.

Transformación dos datos de SO_2

Por último, para o caso do dióxido de xofre, seguimos a mesma lóxica que nos casos anteriores. Os nomes dos mapeos das variables atópanse en [17]. Aplicamos sobre os datos que importamos as operacións:

1. `SO2_column_number_density_validity 50`: neste caso, o manual do dióxido de xofre ([21]) indícanos que debemos considerar os datos cun valor de qa_value maior que 0.5.

2. `derive(O3_column_number_density [Pmolec/cm2])`: derivamos a variable que orixinalmente se denomina sulfurdioxide_total_vertical_column das súas unidades orixinais (mol/m^2) a $Pmolec/cm^2$.

3. `keep(latitude, longitude, latitude_bounds, longitude_bounds, SO2_column_number_density)`: incluímos as variables no produto.

Sobre os datos importados, executamos as seguintes operacións:

1. `bin_spatial(2001, 35, 0.005, 3001, -10, 0.005)`: creamos unha cuadrícula de 2001 puntos de alto, comezando no paralelo 35°, cun *offset* de 0.005°;

2. e 3001 puntos de ancho, comezando no meridiano -10°, cun *offset* de 0.005.

3. `exclude(latitude_bounds_weight, longitude_bounds_weight, weight, latitude_weight, longitude_weight)`: excluímos estas variables do produto final.

4. `derive(latitude{latitude})` e `derive(longitude{longitude})`: derivamos as coordenadas de latitude e lonxitude da nova cuadrícula.

As post-operacións que se aplican son as mesmas que nos anteriores casos. O arquivo resultante almacénase no directorio indicado, e o seu nome segue a expresión `SO2_AAAAMM`, onde AAAA é o ano e MM é o mes o arquivo que exportamos.

3.4. Implementación do sistema

Unha vez comprobado que tódolos scripts funcionan correctamente, comezamos coa implementación do sistema deseñado na máquina de AWS que actuará como servidor. Os pasos a seguir serán configurar por completo a máquina, instalando todo o software e paquetes precisos; instalar e configurar o servidor ERDDAP, para que lea os datos que se almacenan na máquina; e a implementación da interface á que accederán os usuarios.

3.4.1. Configuración da máquina de AWS

Como se comentou anteriormente, empregaremos os Servizos Web de Amazon para configurar unha instancia de EC2 e empregala como servidor, instalando nela o servidor ERDDAP e procesando os datos. Desta forma, facilítase que calquera usuario poida acceder dunha forma sinxela á aplicación dende a web.

A instancia que se elixiu é de tipo t3.xlarge, que dispón de 4 CPUs virtuais con 16 GiB de memoria, cunha AMI (Amazon Machine Image) de Ubuntu Jammy 22.04. A esta instancia acoplóuselle un volume de almacenamento SSD de tipo gp2, con 250 GB de almacenamento, que permite 750 IOPS (*Input/Output Operations per Second*). Ademais, obtívose unha dirección IP elástica, que se asociou

á máquina, de forma que garantizamos que a dirección IP non vai cambiar co tempo, e a aplicación será accesible sempre dende a mesma dirección. Esta dirección IP é a `13.53.65.240`. Todos estes recursos están asignados á rexión eu-north-1 de Europa (Estocolmo), na zona de dispoñibilidade eu-north-1a.

Para poder acceder correctamente á máquina, debemos configurar as regras de entrada e saída do grupo de seguridade asociado á mesma. Neste caso, configuramos as regras de saída para permitir o tráfico HTTP e HTTPS (portos 80 e 443) a tódolos destinos. No que ás regras de entrada se refire, permitimos tamén a entrada de todo o tráfico HTTP e HTTPS (portos 80 e 443); o tráfico SSH (porto 22), tamén dende tódolos orixes, para poder conectarnos remotamente á máquina empregando unha terminal; e o tráfico TCP personalizado ó porto 8080, xa que é o porto que emprega Tomcat, onde estará o servidor ERDDAP e a aplicación web.

Unha vez que temos a máquina funcionando, podemos instalar todo o software que será preciso para executar a aplicación. En primeiro lugar, instalamos Anaconda, unha distribución de Python que simplifica a xestión e instalación de paquetes. A súa instalación é moi sinxela, pois só é preciso descargar o instalador para ubuntu e executalo, seguindo os pasos que indica. Unha vez instalado Anaconda, crearásenos un entorno, que será dende o cal descarguemos os paquetes e executemos a aplicación. A lista de paquetes completa atópase no anexo A. A continuación, debemos instalar Java, na versión 17. Debemos empregar esta versión, ou superior, para poder configurar correctamente o servidor ERDDAP.

O seguinte paso será instalar e configurar Tomcat. Debemos crear un usuario específico, `tomcat`, sen privilexios, para facer máis segura a instalación. Posteriormente, descargamos e extraemos o *core build* de Tomcat nun directorio (que, no noso caso, será `/opt/tomcat`), sobre o cal podemos conceder permisos ó usuario tomcat. Tamén debemos configurar os administradores, para poder acceder ás páxinas de *manager* e *host-manager*. O último paso será crear un servizo en `systemd`, para que Tomcat poida executarse en segundo plano. Nel, configuraremos as variables de entorno coma JAVA_HOME, JAVA_OPTS, CATALINA_BASE e CATALINA_HOME. Recargamos systemd e xa poderemos iniciar o servizo de Tomcat, o cal permitiremos que se execute co inicio do sistema. Por último, debido a que Tomcat emprega o porto 8080, debemos permitir o tráfico nese porto tamén dende a máquina, empregando o comando `sudo ufw allow 8080`.

Neste punto, empregando os scripts que deseñamos en 3.3, executamos o módulo principal (`app.py`), decargando os datos para os catro parámetros que teremos dispoñibles para un período de tempo razonable. No noso caso, decidimos descargar tódolos datos do ano 2023, dende xaneiro ata decembro. Posteriormente, este rango temporal poderase ampliar.

Finalmente, con Tomcat executándose e os datos dispoñibles no servidor, podemos comezar coa configuración do servidor ERDDAP.

3.4.2. Configuración de servidor ERDDAP

Antes de comezar a instalar o servidor ERDDAP, debemos facer perquenos cambios nos arquivos de configuración de Tomcat. No arquivo `server.xml` (*/opt/-tomcat/conf/server.xml*), debemos cambiar o valor do parámetro `connectionTimeout` dentro da etiqueta $< Connector >$ do porto 8080. No arquivo `context.xml` (*opt/-tomcat/conf/context.xml*), debemos cambiar o valor de `cacheMaxSize` da etiqueta $< Resources >$ a 80000.

A continuacion, debemos descargar dende un repositorio de GitHub, proporcionado en [3], un arquivo zip cos arquivos de configuración de ERDDAP. Debemos descomprimir este arquivo na carpeta de Tomcat, creando o directorio */opt/tomcat/content/erddap*. Dentro dos arquivos desta carpeta, atopamos `setup.xml`, o arquivo coa configuración que especifica como se comporta o servidor ERDDAP. Nel, debemos cambiar: $< bigParentDirectory >$, que no noso caso establecemos como */home/ubuntu/erddap/*; $< baseURL$, que é a URL dende a que se accederá ó servidor; $< emailEverythingTo >$, que é a dirección de correo á que se envían mensaxes de erro; e tódolos campos con etiquetas $< email* >$ e $< admin*$, con información relativa ó administrador do sistema (por exemplo, $< adminIndividualName >$, co noso nome; $< adminCity >$, coa cidade; ou $< adminEmail >$, co noso email).

Con isto listo, podemos instalar o arquivo war de ERDDAP. Para isto, podemos descargalo directamente dende a URL que se indica en [3], descargándoo directamente no directorio */opt/tomcat/webapps*. Con isto, teríamos configurado xa o servidor ERDDAP cunha instalación básica, con tódolos datasets que veñen por defecto. Este está accesible na url `http://13.53.65.240:8080/erddap/index.html` . Para poder incluír os nosos propios datasets, deberemos modificar o arquivo */opt/tomcat/content/erddap/datasets.xml*, seguindo os consellos que se indican en [4]. Os creadores de ERDDAP proporcionan, ademais, unha ferramenta denominada `GenerateDatasetsXml`, o cal permite facilitar este proceso.

Para cada datasets, crearemos unha etiqueta $< dataset >$, na cal configuraremos parámetros coma o *type* (EDDGridFromNcFiles), o *datasetId*, o seu estado (*active=true* ou *active=false*), cada canto se recarga e se actualiza ($< reloadEveryNMinutes >$, que estableceremos a 10080 (7 días); e $< updateEveryNMillis >$, que será 60000 (1 minuto)), o directorio onde se atopan os arquivos (no caso de, por exemplo, o monóxido de carbono, será */home/ubuntu/TFG/data/CO/*), ou a expresión regular no nome dos arquivos (que é *.*\.nc*) . Podemos engadir atributos, coma o nome título do dataset, un resumo dos datos que ten ou as variables do mesmo. Tamén engadiremos tres `axisVariable`: unha para o tempo, que podemos derivar do nome do arquivo, xa que este é unha expresión regular; a latitude, que se obtén directamente do arquivo; e a lonxitude, que tamén se obtén do arquivo. Por último, engadiremos unha `dataVariable`, que será a variable que se mostrará no mapa. Engadirémoslle atributos coma o tipo de dato (*double*), o nome, o valor mínimo e o valor

máximo. A configuración completa do arquivo datasets.xml está no anexo A.

3.4.3. Realización da interface

Para crear a interface, crearemos unha aplicación nova en Tomcat, para illala do servidor ERDDAP. Crearemos unha nova carpeta no directorio */opt/tomcat/webapps*, denominada TROPOMIVisor. Este será o nome da aplicación, que teremos que poñer na URL para acceder á mesma. Desta forma, a url para acceder á aplicación sería `http://13.53.65.240:8080/TROPOMIVisor/`

Capítulo 4

Probas

Capítulo 5

Conclusións e posibles ampliacións

Apéndice A

Manuais técnicos

En función do tipo de Traballo e metodoloxía empregada, o contido poderase dividir en varios documentos. En todo caso, neles incluirase toda a información precisa para aquelas persoas que se vaian encargar do desenvolvemento e/ou modificación do Sistema (por exemplo código fonte, recursos necesarios, operacións necesarias para modificacións e probas, posibles problemas, etc.). O código fonte poderase entregar en soporte informático en formatos PDF ou postscript.

Apéndice B

Manuais de usuario

Incluirán toda a información precisa para aquelas persoas que utilicen o Sistema: instalación, utilización, configuración, mensaxes de erro, etc. A documentación do usuario debe ser autocontida, é dicir, para o seu entendemento o usuario final non debe precisar da lectura doutro manual técnico.

Bibliografía

[1] Data Products - Sentinel-5P Mission - Sentinel Online. *Sentinel Online.* [en liña] [sen data] Dispoñible en *https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-5p/data-products*.

[2] ERDDAP - Home Page. *National Centers for Environmental Information (NCEI).* [en liña] [sen data] Dispoñible en *https://www.ncei.noaa.gov/erddap/index.html*

[3] Set Up Your Own ERDDAP. *ERDDAP.* [en liña] [sen data] Dispoñible en *https://coastwatch.pfeg.noaa.gov/erddap/download/setup.html*

[4] Working with the datasets.xml File. *ERDDAP.* [en liña] [sen data] Dispoñible en *https://coastwatch.pfeg.noaa.gov/erddap/download/setupDatasetsXml.html*

[5] ARSET - High Resolution NO2 Monitoring From Space with TROPOMI. *NASA Applied Sciences.* [en liña] [sen data] Dispoñible en *https://appliedsciences.nasa.gov/get-involved/training/english/arset-high-resolution-no2-monitoring-space-tropomi*

[6] ARSET - Data Analysis Tools for High Resolution Air Quality Satellite Datasets *NASA Applied Sciences.* [en liña] [sen data] Dispoñible en *https://appliedsciences.nasa.gov/get-involved/training/english/arset-data-analysis-tools-high-resolution-air-quality-satellite*

[7] ARSET - Applications of Satellite Observations for Air Quality and Health Exposure *NASA Applied Sciences.* [en liña] [sen data] Dispoñible en *https://appliedsciences.nasa.gov/get-involved/training/english/arset-applications-satellite-observations-air-quality-and-health*

[8] Sentinel-5P L2. *Sentinel Hub.* [en liña] [sen data] Dispoñible en *https://docs.sentinel-hub.com/api/latest/data/sentinel-5p-l2/*

[9] S5P Mission. *SentiWiki Home.* [en liña] [sen data] Dispoñible en *https://sentiwiki.copernicus.eu/web/s5p-mission#TROPOMI-Response-Functions-(Acquisition-Modes)*

[10] Types of air pollution. Which pollution is the most dangerous? — Air Quality Monitoring. Monitor in UK & Europe. Airly Data Platform and Monitors. *Air Quality Monitoring. Monitor in UK & Europe. Airly Data Platform and Monitors — Airly.* [en liña] [sen data] Dispoñible en *https://airly.org/en/the-most-dangerous-types-of-air-pollution/*

[11] Documentation - OData. *Copernicus Dataspace* [en liña] [sen data] Dispoñible en: *https://documentation.dataspace.copernicus.eu/APIs/OData.html*

[12] *Documentation - Quotas and Limitations.* Documentation [en liña] [sen data] Dispoñible en: *https://documentation.dataspace.copernicus.eu/Quotas.html*

[13] HARP 1.20.2 documentation *HARP manual* [en liña] [sen data] Dispoñible en: *https://stcorp.github.io/harp/doc/html/index.html*

[14] S5P_L2_CO — HARP 1.20.2 documentation. *HARP manual* [en liña] [sen data] Dispoñible en: *https://stcorp.github.io/harp/doc/html/ingestions/S5P_L2_CO.html*

[15] S5P_L2_NO2 — HARP 1.20.2 documentation. *HARP manual* [en liña] [sen data] Dispoñible en: *https://stcorp.github.io/harp/doc/html/ingestions/S5P_L2_NO2.html*

[16] S5P_L2_O3 — HARP 1.20.2 documentation. *HARP manual* [en liña] [sen data] Dispoñible en: *https://stcorp.github.io/harp/doc/html/ingestions/S5P_L2_O3.html*

[17] S5P_L2_SO2 — HARP 1.20.2 documentation. *HARP manual* ([en liña] [sen data] Dispoñible en: *https://stcorp.github.io/harp/doc/html/ingestions/S5P_L2_SO2.html*

[18] Sentinel-5P TROPOMI Level 2 Product User Manual - Carbon Monoxide. *Sentinel Online* [en liña] [sen data] Dispoñible en: *https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-5p-tropomi/document-library/-/asset_publisher/w9Mnd6VPjXlc/content/sentinel-5p-tropomi-level-2-product-user-manual-carbon-monoxide*

[19] Sentinel-5P TROPOMI Level-2 Product User Manual - Nitrogen Dioxide. *Sentinel Online* [en liña] [sen data] Dispoñible en: *https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-5p-tropomi/document-library/-/asset_publisher/w9Mnd6VPjXlc/content/sentinel-5p-tropomi-level-2-product-user-manual-nitrogen-dioxide*

[20] Sentinel-5P TROPOMI Level 2 Product User Manual - Ozone Total Column. *Sentinel Online* [en liña] [sen data] Dispoñible en: *https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-5p-tropomi/document-library/-/asset_publisher/w9Mnd6VPjXlc/content/sentinel-5p-tropomi-level-2-product-user-manual-ozone-total-column*

[21] Sentinel-5P TROPOMI Level 2 Product User Manual - Sulphur Dioxide SO2. *Sentinel Online* [en liña] [sen data] Dispoñible en: *https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-5p-tropomi/document-library/-/asset_publisher/w9Mnd6VPjXlc/content/sentinel-5p-tropomi-level-2-product-user-manual-sulphur-dioxide-so2*