Workflow Engine for Earth Observation Services

Diogo Rafael Lopes Ferreira
Instituto Superior Técnico, Universidade de Lisboa

Abstract—Nowadays, satellites are gathering massive amounts of remote sensing data from planet Earth. Scientists transform this data through specialized tools (GDAL, SNAP, and QGIS) to forecast future events and monitor Earth's current health. As a matter of fact, more than one transformation might be required, and it may need a significant amount of resources, such as CPU and bandwidth.

Given these requirements, the advances of the Internet, and the increase in the sharing of resources, web services became an attractive solution to tackle them. So, local tools, in EO, started to be exposed as WPSs (Web Processing Services) - a specialized standard created by the Open Geospatial Consortium (OGC).

Therefore, we want to be able to chain WPSs, i.e., create a workflow. Hence, three fundamental problems emerged: how to chain, validate and deploy workflows of WPSs. Generally speaking, the solution to all these problems can be found in a Workflow Management System (WMS).

At the moment, WMSs from other fields are being used to fulfill the EO needs. Nonetheless, this comes at the expense of losing the expressiveness of the WPS standard, key to the validation process, and the interoperability between the other OWS (OGC web services).

Given these points, we propose in this work a community-driven solution to assure interoperation between WPS services, through syntactic validation of inputs and outputs during the workflow modelling and runtime phases. Along with, the adoption of JSON as the validation and workflow representation language; a modeler to compose and automate the creation of workflows; the possibility to translate the internal workflow representation to any target WMS language.

Index Terms—Earth Observation, WPS, Workflow, Composition, Validation, Translation.

I. INTRODUCTION

Earth Observation (EO) is the gathering of information (physical, chemical and biological) about planet Earth through the use of remote sensing technologies. Remote sensing is the science of obtaining information without physically being in contact with it [1]. Satellites are a particular use case of such technology, since they use their sensors to monitor the Earth.

Nowadays, EO data is available to the public, and it can be used to monitor the effects of natural catastrophes, human civilization, diseases propagation, and climate change. In addition, it can be used in many different fields such as agriculture (precision farming), food security, health and air quality, ecosystems, and oceans [1]. Thus, by collecting this data, we are more apt to devise measures to prevent future events and mitigate current ones.

With such variety of uses, EO data is raising interest among different areas (scientific, social, economic and political) and sectors (public or private, research or applications). This became possible, mainly, due to the disclosure of remote sensing data, advances in technology, and new market opportunities.

A. Problems

There are large amounts of geospatial data being gathered every day that require processing through the chaining of WPSs, which are, predominantly, resource intensive, deployed on the cloud, to extract usable information. Thus, our focus will be to guarantee the successful chaining of WPSs. In other words, we want to be able to create workflows of WPSs. However, two problems emerge:

- How do we automate the workflow creation, allow its reusability and its deployment to the cloud with confidence\(^1\) to avoid waste of resources, such as allocation of machines on the cloud under a pay-per-use model?
- How and where do we execute the workflow?

B. Outline

This document is divided into five sections. Background, II, where we present the foundations of our solution, i.e., the current work being performed in EO regarding workflows. Implementation, III, which describes our solution in detail. Results, IV, that proves our approach through a real-world example of an EO workflow. Conclusion, V, where we summarize our achievements, and Future Work, VI, in which we give our personal opinion on the direction of future studies.

II. BACKGROUND

The purpose of this section is to explore the field and its current state of the art, in order to devise a solution to the problems in I-A.

A. Web Processing Service (WPS)

The WPS standard abstracts local tools and algorithms that manipulate EO data remotely, through processes. It goes without saying, that we need to understand the standard to grasp possible validation problems between WPSs, perceive how an engine could exploit them, how they interact with each other, etc. Thus, to find if two WPSs are valid, i.e., if they can be chained together, we need to know which are the inputs/outputs and operations that the standard defines.

\(^1\)Confidence here means that the chain represents a valid execution.
1) **Data Types:** WPS\(^2\) defines three types of I/O:

- **LiteralData:** encodes atomic data such as scalars, linear units, or well-known names. Domains for LiteralData are a combination of data types (e.g. Double, Integer, String), a given value range or allowed values, and an associated unit (e.g. meters, degrees Celsius);

- **ComplexData:** are usually raster or vector files, which can be served as a reference or locally\(^3\). It specifies the expected file mime type (e.g. application/gml+xml), the schema (e.g. xsd) that the file must comply, and the awaited encoding (e.g. utf-8). Besides its specification, its purpose is to abstract EO products (spatial data);

- **BoundingBox:** is data that serves a variety of purposes in spatial data processing, usually coordinates in the form of an array that represents an area. Some simple applications are the definition of extents for a clipping operation or the definition of an analysis region.

Furthermore, every data type needs to define extra properties: **identifier**, **title**, **abstract**, **minOccurs** and **maxOccurs**. In addition, depending on the data type, it can be defined more optional attributes, i.e., in a LiteralData, you can define the Unit of Measure (UOM), as per example meters for the distance metric system. Note that all of these should be considered to perform the correct validation of a WPS.

2) **Operations:** WPS provides three important operations, through HTTP Get and Post methods, according to [2], which are: **GetCapabilities** - lists all of the processes available in a WPS; **DescribeProcess** - allows the user to request a full description of a process, i.e., its metadata and I/O; **Execute** - triggers the execution of a process given its required inputs. Note that the response of every operation is a xml document, and each WPS is composed of at least one process. Moreover, processes, similarly to data types, need to define the following attributes: identifier, title, and abstract. Along with that, they also need to describe the list of inputs and outputs, which should comply with the properties presented in II-A1.

**B. Workflow Validation**

A workflow chain has many links with different characteristics, being one of those the type of inputs and outputs of each process. Traditionally, workflow modelling, in EO, is focused on its structural aspects, mainly, the order of execution, the orchestration, and how to express it through a language [3]. Thus, currently very little is done on the syntactic validation of workflows.

1) **Syntactic Validation:** serves to know if the model will, at least, run in a Workflow Management System (WMS). Note that it only provides means to check for potential problems within the workflow [3], by weeding out incompatible services. Thus, validating it does not imply that errors will not occur during runtime. Furthermore, it helps, similarly to compilers in programming languages, to validate larger workflows quickly and with precision. In fact, analogously to compilers, we need a language to express the workflow (data flow, data types, and constraints), in order to validate it. As a result, the main goals behind syntactic validation of an EO workflow are providing confidence in remote deployments and automate their creation. Hence, by fulfilling both, we improve the rate of spatial data consumption. Surely, syntactic validation will introduce some overhead, since we need to validate the workflow instance before executing it.

2) **Data Types & Source:** not all data required by a workflow is generated by the processes that compose it. Thus, we can distinguish two sources of data, internal (generated in the workflow) and external. As a result, this means that syntactic validation can be done at two phases, **build** and **runtime**.

3) **Data Validation Problems:** may rise from syntactically validating the workflow. According to [3], there are four types of problems, which are introduced in table I.

<table>
<thead>
<tr>
<th>Validation Problems</th>
<th>Brief description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mismatching data</strong></td>
<td>Input types of the next process, in the chain, do not match the output types of the previous process. Or the input filled by the user does not match the required type.</td>
<td>X is required as input for the process B, but A, a previous process, has generated Y different from X.</td>
</tr>
<tr>
<td><strong>Missing data</strong></td>
<td>Not enough data was generated by the previous activities to guarantee the execution of an process. Users must express the missing data.</td>
<td>X is required as input for the process B, but i.e., neither has X been manually introduced by the user or generated by the previous process.</td>
</tr>
<tr>
<td><strong>Redundant data</strong></td>
<td>Output data is generated in “excess” from the previous process and is not required in the following process.</td>
<td>X is generated by A, but is not required by the following process, say B, in order to execute, thus X can be considered redundant.</td>
</tr>
<tr>
<td><strong>Insufficient data</strong></td>
<td>Unavailable services, lack of understanding of the meaning of each data type and its usage, or exceptions running activities.</td>
<td>X is required as input for the process B, but the previous process, A, is not responding due to service unavailability.</td>
</tr>
</tbody>
</table>

**TABLE I: Data validation problems.**

**C. Workflow Management Systems (WMS)**

It is now clear that the solutions to the problems presented in I-A, are found in WMSs. Thus, studying them will help us build our solution.

1) **Workflow Description Formats:** as mentioned in II-B1, to perform validation, we need to express the workflow through a language. Besides, a description format should be used to deliver the workflow to an engine, and to be able to create templates to reuse it. Thus, we have explored the following languages:

\(^2\)http://docs.ogc.org/is/14-065/14-065.html#25

\(^3\)Given that they are accessible in the context of the WPS execution.
• **BPEL**: as introduced in [4], is in version 2.0, and was released by OASIS. It is used to orchestrate web services, i.e., specifies the executable processes and how messages are exchanged between each other. It was designed to use the Web Service Description Language (WSDL). There is no graphical notation, nonetheless, it is possible to convert BPEL (1.1) to BPMN. It is perhaps the most used description format in the geospatial field [5], [6], [7], and [8];

• **BPMN**: is a newer standard adopted in the business area to model processes. It has a graphical notation, and many workflow execution engines are adopting this standard, since it was approved by the International Organization for Standardization (ISO). Recent studies, [9] and [10], have tried to use it to orchestrate OGC Web Services;

• **DSL**: assumes the creation of our own workflow language. Such example can be found in GiSHEO with the creation of the description format SiLK (Simple Domain Language) where they expressed workflows of OGC Web Services [11]. Clearly, having expressiveness is a plus, because we can adapt the domain to our needs, however this would mean that we would require our own workflow engine capable of interpreting this new language. This would ultimately mean that any kind of interoperability between other WMSs is lost. Nonetheless, we can translate the DSL to any other language, thus creating the possibility of running the workflow in any engine.

2) **Scientific WMS**: were discussed in [12], where three were considered the most relevant. In fact, none of these were created with EO in mind, instead, researchers from the area saw them as potential candidates to perform EO workflows. Firstly, we have Pegasus [4] and its validator Wings, followed by Kepler [5], and ending with Taverna [6]. Note that the last two had their own attempts to adapt the WPS standard, respectively, in [13] and [14].

3) **Business & Cloud WMS**: are becoming popular in the geospatial field, and are part of case studies, such as engineering reports [4] and testbeds [15]. Thus, we have explored the most significant business solutions applied to EO, jBPM [7], Amazon Simple Workflow Service [8], and Camunda [9].

In sum, we have concluded that there no solution will fulfill the demands of every field, and if there was, it would eventually diverge.

## III. IMPLEMENTATION

During section II, we have introduced all of the fundamental concepts and different approaches to deal with the problems in I-A. Consequently, we will now discuss its implementation, by focusing on its architecture.

### A. Modules

We have two core modules, the validation module (WPS-V) and the web module (WPS-V-WEB), where the latter uses the first. We will now separate the problem solving, I-A, between the modules:

- **WPS-V**: is responsible for providing validation of EO workflows;
- **WPS-V-WEB**: has a canvas available to compose workflows of WPS in the web; suggests/alerts the user of compatible processes in the chain, i.e., processes that share outputs-inputs; defines the types and restrictions on inputs that the users need to follow; provides complete error messages about incompatibilities during the workflow composition process; allows the user to download workflow templates and import them; enables the user to execute the workflow. As a result, the web module clearly solves the automation, reusability, and execution problems.

### B. WPS-V

As seen in II-B, to provide syntactic validation on EO workflows, we need to take into account the description formats (BPMN, BPEL, DSL), II-C1, the data types and their source (internal, external), II-B2, the phase of validation (build time and runtime), II-B1, and lastly the data validation problems (mismatching, missing, redundant, insufficient), II-B3.

1) **Syntactic Validation - Description Formats**: from II-C1 we have concluded that both the WPS standard and the OGC stack, are too specialized to be adopted by any already existing language for validation purposes. Since, BPEL and BPMN do not provide enough expressiveness to encapsulate the WPS standard, due to the mismatching of types of data, operations, etc. Nevertheless, if the problem was only execution, they would make perfect sense. Hence, building our own DSL is our best option, since it provides freedom and flexibility, when expressing workflows of WPS. We opted to create our workflow specification in JSON, since it is easy to read (human-readable), REST ready, and effortless/quick to verify against a JSON schema.

2) **Syntactic Validation - Build Time and Runtime**: is related to the source of data, II-B2. Clearly, the first can only check inputs provided by the user (external inputs) and the redirection of outputs between processes, i.e., the usage of an output of a previous process by another one (internal inputs). However, inputs generated by the workflow, in the build time, are only verified at a specification level, which means that they might be compatible, but until the output is generated, we are not completely certain of it. As a result, we provide syntactic validation at both phases.

### C. Syntactic Validation - Validation problems

Taking into account the WPS standard and section II, we distinguish four levels to accomplish syntactic validation in geospatial workflows, which can be seen in figure 1. Analogously to a staircase, to reach the next step we should complete the previous one, hence to be able to validate at level 1, we

---

4https://pegasus.isi.edu/
5https://kepler-project.org/
6https://taverna.incubator.apache.org/
7https://docs.jboss.org/jbpm/release/7.11.0.Final/jbpm-docs/html_single/
8https://aws.amazon.com/swf/
9https://camunda.com/
need to ensure that level 0 is satisfied. Being said that, we will
now "climb the staircase".

Fig. 1: Levels of syntactic validation in EO workflows.

1) Level 0: checks if the DescribeProcess of a WPS process
specification (xml) obeys the schema (xsd). Note that it would
not make sense trying to validate something that does not obey
the standard.

2) Level 1: given an input, validates it against its speci-
cation/restrictions, thus, solving the mismatching of data. In
fact, each type of data has its own validation problems, as seen
below:

- **LiteralData**: data type, range, allowed values;
- **BoundingBox**: coordinate reference system;
- **ComplexData**: mime type, encodings, schema, maxi-
mum file size.

Clearly, there are many constraints for each type of data.
So, how do we efficiently verify the inputs against all of these
constraints? The answer to this problem can be found in 2,
which was based on [16].

From figure 2, it is evident that there are four types of
validators, such that three of them correspond to our types
of data, having each one of them its own constraints. In
addition, we have a Process Validator, which is composed of
Validators, thus for every input in our process specification we
are creating only one validator and reusing it for each call of
isValid with the correct inputDto. This makes the verification
a homogeneous procedure, where at the process creation, we
attach a Process Validator that will delegate the validation to
the correct validator.

Fig. 2: The structure of the different types of validators for
Level 1.

Regarding the constraints, it should be noted that each has a
specific type, corresponding to the type of data restrictions that
they abstract. These are constructed by receiving the domain
specification for an input, per example, in a BoundingBox-
Constraint it receives a BoundingBox, where it builds the
verification context for supporting future calls to the isValid
method. The inner works for a BoundingBoxConstraint can be
found in figure 3.

A BoundingBox has a list of supported CRS’s, from where
we extract every CRS into a map of CRS handlers, where
the key is the CRS identifier, such as EPSG:4326. The
CRSHandler is a simple wrapper of Apache SIS [10], which
provides us with a pool of CRSs with their respective longitude
and latitude bounds. Thus, the isValid method is as simple as
algorithm 1.

Fig. 3: A BoundingBoxConstraint has into account the val-
ification problems mentioned, as seen by the methods names,
which are self explanatory. However, note that a bounding box
input specification may support more than one CRS, thus the
field crsHandlers.

Algorithm 1: BoundingBoxData Validator pseudo-
docode.

\[
\text{input} \leftarrow \text{BoundingBoxDto};
\]

\[
\begin{align*}
\text{if} \ crs \in \text{getCrshandlers} \ & \ \text{then} \\ \\
& \text{if} \ \text{input.longitudeBound} \in \text{getLongitudeBound} \ \text{and} \\
& \ \text{input.latitudeBound} \in \text{getLatitudeBound} \ \text{then} \\
& \quad \text{return} \ \text{true;} \\
& \quad \text{else} \\
& \quad \quad \text{throw new ValidationException;} \\
& \quad \text{end if} \\
& \text{else} \\
& \quad \text{throw new ValidationException;} \\
& \text{end if}
\end{align*}
\]

It is easily extrapolated, based on the validation problems,
how we could construct the other constraints. As a clue, Com-
plexDataConstraints support themselves on Apache Tika11,
which is capable of validating mime types and encodings.
Both MaximumFileSize and Schema can easily be verified.
In fact, the last was already discussed in III-C1. Moreover,
LiteralDataConstraints use Google Guava12 to define Ranges,
and Java XML DataTypeConverter to verify Data Types against

10http://sis.apache.org/
11https://tika.apache.org/
12https://github.com/google/guava
the specified type, this is, we try to convert the inputDto value into the target defined type.

Thus, this way of validating data against its specification is extremely intuitive as shown above. In conclusion, constraints built the restrictions to a given domain specification, which are called by the isValid method that compares the input against them.

3) Level 2: considers the workflow specification, written in JSON, verifies it against the schema and imports it to a domain object after unmarshalling. So, achieving it solves the missing, redundant and insufficient data, since we are looking at the workflow requirements. Therefore, in this level we validate internal inputs compatibility, and Minimum & Maximum Occurrences.

The solution to level 2 can be seen at figure 4. It displays, at runtime, the validation and creation of a workflow. Note that validation is provided in two ways, at creation and as a separate interface, this is, for workflows that are iteratively created in the domain, instead of an external source, such as modeler.

To create a workflow, we start by receiving a specification in JSON at the createWFAsJson interface, provided by the WorkflowHandler instance. There, at the parseWF, we send the JSON representation of the workflow to a JsonValidator instance, to verify it against the Workflow JSON Schema. If it is valid, then we proceed to parsing the JSON workflow to a domain object, by exploiting the provided interface, fromJson, in Gson. Lastly, we will send the workflow object to WFValidator, which was constructed based on figure 2.

Fig. 4: Level 2 Component and Connector diagram explaining the runtime behaviour when creating and validating an workflow.

4) Level 3: verifies that the outputs generated within the workflow and that are being used by other processes obey the specification. Its implementation is provided at the end of this section.

D. WPS-V-WEB

This module was created to fulfil the requirements and functionalities of EO. In fact, this frontend is based on the characteristics of the WMSs introduced in II-C, as we can not use any of them, since we have already chosen our own DSL. Yet, engineering taught us to reuse previous work. So, with that in mind, below, we present the module implementation.

1) Workflow Creation Automation: is provided by the possibility to compose workflows via a graphical interface. GUIs, surely, help to intuitively build workflows. Thus, we have built a canvas, figure 5, where the user can drag-and-drop WPS processes and chain them together. Its palette is based on an extremely small subset of the BPMN graphical specification, only providing the user start events, service tasks, and end events.

Fig. 5: WPS-V-WEB diagram canvas.

In addition, this web module helps the user fill missing inputs, by showing only the relevant information to satisfy it. Besides, it is also able to supply the user multiple, i.e., the possibility to reuse an output from a previous process, and choose to fill or not an optional input. Regarding the multiple inputs, we added a functionality that deviates from the WPS standard but favours the practicability of the GUI, since an input may have a large number of maximum occurrences, we decided to let the user import the inputs through a .csv file.

Finally, and equally important, the WPS-V-WEB module, displays the error messages that rise from the workflow validation in the WPS-V module, III-B. These are presented in a console, similarly, to compilers, figure 6. Furthermore, according to the validation result, either succeeding or failing, it paints, in the correspondent colour (red or green), the links that have problems, to ease the work of the user to correct it. Note that, if you click in the error you get redirected to the link, which has the unconformity.

2) Workflow Reusability: is solved by using templates. Given that we have a graphical representation of the workflow, we will need to save it as well. Both, graphical and workflow, are saved in JSON, that can be downloaded by clicking in the Download Template, figure 5. Obviously, the reverse can also be done, i.e., imported.

3) Workflow Execution: is solved through the translation of our language. To clarify, we began this section by stating that we could not use any WMS, since validation led us to a path where we were forced to choose a DSL. Nonetheless, a
practical solution was found to tackle this problem. Although WPS-V was the one that caused it in the first place, funny enough, its existence provides a better solution, that does not rely on a language or an engine. The reasoning behind it is evident, i.e., we control the composition and the validation flow, therefore, we can translate our DSL to any other workflow language, such as BPEL or BPMN being sure that it will run. Hence, we have found a middle ground, where we can still use every WMS, with all of its features, and validate the chain of WPSs.

### E. Runtime Validation

Given that both modules are introduced, at this point, we can explain how we have solved runtime validation. Succinctly, the problem is that we lose control over the validation flow, when the workflow starts to run. Therefore, since we do not control the WMSs, the only thing that we can do is to exploit the natural way that they operate in order to return control to our application. But, what do we mean by the “natural way of operate”? Well, they, fundamentally, manage and run workflows composed by processes, that are executable processes or web services. Therefore, we have added processing elements that call our application through a rest endpoint, listing 1, without user innervation.

```
@PostMapping(value = "/validate/{workflowID}/[index]"
) public void validate(HttpServletRequest httpServletRequest, 
  @PathVariable("workflowID") String workflowID, 
  @PathVariable("index") String index) { 
  linkValidatorService.validate(httpServletRequest, workflowID, 
    Integer.valueOf(index));
}
```

Listing 1: Spring Rest Endpoint for regaining control of the validation flow. The index corresponds to the link we are trying to validate according to its order in the chain.

Take into account that every language will have these processing elements, and tweaks must be made during the translation process to perform a rest call. As a result, translating to a WMS that does not support web services (HTTP calls), will not work. Also, the `httpServeletRequest` parameter serves to pass the Execute Response to extract the resulting outputs.

In sum, it can be seen, in figure 8, our solution. Besides providing runtime validation, we can use this to monitor the workflow execution, since we are always aware of its stage. Equally important, we are capable of, when a runtime validation error occurs, give back the control to the user, by implementing the `abort()` method in 7, which can be used to, per example, abort the execution gracefully, i.e., turn down instances. Additionally, you might decide that you want to allocate resources, only when you are sure that the next process is a valid execution, does saving possible waste of resources.

As expected, our solution to validate the workflow at runtime has associated costs. These affect, primarily, the performance, due to the increase in the number of communication events. We have found that runtime validation increases the number of communication events by \(2 \times L; \text{if } L > 1\). Moreover, we are forcing the outputs of a process to do extra hops to be validated, which, ultimately, in EO depending on the workflow composition, will result in passing through the network big files (gigabyte), hence adding a lot of overhead to the workflow execution.

### IV. RESULTS: SENTINEL-2 IMAGERY TO IDENTIFY AND MAP WILDFIRE EVENTS

In this chapter, we aim to prove our solution. Therefore, we present a real-world use case of WPSs to identify and map wildfire events. It should be noted that this example is based and inspired on a Copernicus workshop, presented by Mallon Technology Ltd.

Out of control wildfires cause extreme long-term damage to the environment, wildlife, flora and property including forestry and agricultural holdings every year.

Along with improving the detection and response times to such fires, there is also a need to improve a post-event delineation, assessment and monitoring of the affected areas. Such post-event analysis can then feed back into strategies.

---

14The process to parse them is just as III-C1.
15https://www.mallontechnology.com/
and policies for wildfire prevention, prediction, mitigation and response.

Systems for detecting wildfires and monitoring the risk of wildfire development, such as EFFIS (European Forest Information System) and AFIS (Advanced Information System), provide excellent up-to-date information on wildfires. However, the detection of such fires by these systems is prone to inaccuracy in terms of the exact location and extents of wildfire events and burned areas, and fail to pick up many of the smaller wildfires, which occur and impact the environment and local communities in a variety of ways.

Entities such as national firefighting units, police departments, environmental protection agencies, civil protection units, forest and agriculture management organisations, national park organizations, farmers, insurance companies, and wildfire interest groups can benefit from this workflow process. Mainly, they would gain more accurate post-event delineation of wildfire extents, since smaller areas affected by wildfires can be determined and assessed, thus, the need for ground surveys on often difficult terrain to determine wildfire locations and extents can be potentially reduced or made easier.

A. Process Steps

The identification and mapping of wildfire events follow the following steps:

1) Access the Sentinel Scihub\textsuperscript{16} (data-discovery) to identify and download appropriate product, level C-1 imagery (data access);
2) Resample the bands according to a reference band in order to have all of them with the same pixel resolution, preferably a 10 meter pixel resolution band (pre-processing);
3) Define an area of interest to create a subset, this is, the area that we want to analyse (pre-processing);
4) Remove cloud coverage on an area of interest, using the administrative boundaries and cloud mask (pre-processing);
5) Apply the burnt ratio formula (processing);
6) Switch the bands to highlight the burnt areas (post-processing).

Note that the second step is needed due to the large size of the sentinel-2 imagery products, hence we are required to subset the image to an area of interest to be able to work efficiently, i.e., cut the processing time. Furthermore, all of these steps require a lot of processing time specially the resampling, which in a machine with 16GB of Ram, Core i7 fourth generation (4 cores, 8 threads, clocking at 2.40GHz), may take up to 18 minutes for a 2GB image of the small area seen in figure 10.

Regarding the fourth step, it is usual that sentinel imagery has cloud coverage, which may impact the assessment of burnt areas. As a result, we do a vector difference between the cloud mask and the administrative boundaries\textsuperscript{17} to identify and eliminate the clouds.

\textsuperscript{16}https://scihub.copernicus.eu/dhus/
\textsuperscript{17}Delineate the land boundaries of, per example, a country.

B. Workflow Schematic

We can build a workflow using the steps found in IV-A, which can be seen in figure 9.

These steps can be translated into web processing services. They will be used to compose a workflow, validate, and translate in order to test our solution. The final result should be compared with the existing data at EFFIS to prove its legitimacy, this is, the identified burnt areas should be the same as the ones registered.

C. Testing the Solution

Now that the scenario/use-case context is explicit, we will apply our solution to solve it based on the topics discussed in III. It should be noted, right from the beginning, that it is impossible to fit every problem in a single use-case, especially the validation problems presented in III-C, since the combination, for example, in the ComplexData would be exponential. However, we will try to cover every data validation problem introduced in II-B3.

The workflow presented in IV-B, can be applied to any region on the world. Thus, we will focus our attention to the fires that occurred in the municipality of Karistos, Greece, north-east of Athens, between mid July and mid August of 2016.

D. Expected Result

The final result should be the same as the one presented in figure 10, which was removed from EFFIS, where the burnt areas are highlighted in red.

Therefore, we consider that our solution is correct if it is able to: compose the workflow of WPSs IV-B, validate the inputs and outputs accordingly to the specification, translate the workflow to be executed in any engine, and, obviously, having the final result matching the one shown.

E. Workflow

The first step to test our solution is to compose the workflow schematic presented in IV-B, as it can be seen in figure 11.

Since the workflow is composed, the next step is to fill in the inputs and outputs. There are two types of inputs according to the source type, internal and external, as it was said in III. These distinguish, respectively, syntactic validation at runtime from the modelling phase, however, when resolved, the internal inputs behave exactly the same as externals in regards to the validation process.
F. Validation Cases

In this subsection, we will demonstrate, for each of data validation problems in II-B3, how they manifest and get solved. Note that to reveal these problems, in order to test our solution, we have used WPSs, which return wrong values.

In this use-case, we can identify cases for mismatching of data for the two of the three types, missing data\textsuperscript{18}, and insufficient data. Additionally, we can also detect occurrences validation. However, we do not have any case of redundant data, which is normal since these WPSs only generate one output. But, even if we had, during III we have clarified that we only validate what is used in the workflow.

1) Mismatching: has to do with the type filled not matching the required, thus we will test this for the bounding boxes and complex data. Note that a test battery was done to test all of the cases presented below and more.

Let's start by the *BoundingBoxData*, at the *subset* input in the Subset WPS, it expects that the value obeys the EPSG:4326 (WGS 84). This means that the CRS must be valid, latitude has to be within the range of $[-180, 180]$, and the longitude between $[-90, 90]$. Hence, the bounding box $[37.918, 38.196, 24.125]$ results in an error, as it can be seen in figure 12. In the same way, by using Apache SIS we are able to support an extensive list of coordinate reference systems, and standards such as WGS 84, NAVD 88, NAD 83, GRS 80, UTM, etc.

\textsuperscript{18}Missing data usually can happen in almost any workflow

\textsuperscript{19}https://sis.apache.org/tables/CoordinateReferenceSystems.html

 Lastly, we have *ComplexData*, which has a considerable representation in the workflow, for instance, we will look at the *boundaries* input in the RemoveCloudCoverage. It expects a vector shapefile representing a polygon of the administrative boundaries of Greece. Note that this process, for the area shown in figure 10, which is considerably small, takes two minutes, so for a bigger area it might take longer, therefore it will be a WPS where we want to dedicate more resources and guarantee a safe execution. Nonetheless, with the composition automation introduced in III, the users are informed of the expected mime type and, thus, they should make fewer mistakes. Even so, errors might occur, so imagine that we have introduced a wrong format file, for example, an image/tiff, our solution picks it up, thanks to Apache Tika, as it can be seen in figure 12.

2) Missing Data: occurs when data is missing to execute the workflow. This type of data validation problem can be replicated almost anywhere in the workflow and is the most simple of them all. So, when the user triggers the validation, the first thing done is the scanning of the workflow for missing inputs, figure 13, to alert the user.

3) Insufficient Data: comprehends the errors during the workflow execution. Usually, it occurs in any output produced that did not obey the specification, i.e., the wrong type of data was produced, or some sort of corruption has happened during its production. Hence, we are dealing with runtime, and errors produced here follow user-specific implementation for the *abort()* method, as seen in 7. Actually, the simplest implementation is to abort the execution of the workflow, as we did for Camunda, through a rest call.

4) Occurrences Validation: validates the number of occurrences of an input. The maximum number of occurrences for
the `sourceBands` at the Subset WPS is 13\textsuperscript{20}, which represents the number of bands in a sentinel-2 product (B1, B2, B3, B4, B5, B6, B7, B8, B8A, B9, B10, B11, B12). So, it should not be possible to input either 0 or 14 bands. Therefore, in the multiple input setup, the user should be warned about this issue, figure 14. Similarly, the WPS-V module also confirms this.

Fig. 14: Number of bands supported in a sentinel-2 image restriction to the user in the multiple input setup.

**G. Translation: Camunda (BPMN)**

As explained in III, the solution to execute workflows of WPS in any engine was to translate it to a target language. Thus, we have decided to translate our internal workflow representation to BPMN, figure 15, in order to be used on Camunda.

Fig. 15: The workflow representation of 11 in Camunda, after the translation process.

It should be noted that we could have chosen any of WMSs presented in II-C. Nevertheless, Camunda provides: rest endpoints for every important control operation, such as deploy and abort, an easy installation, and support for BPMN. Moreover, Camunda is gaining popularity in EO, [4] and [15].

Briefly, the workflow presented above is composed of two tasks, user and service. We can locate them, respectively, at the top and the bottom. Service tasks are responsible for calling the WPS Execute operation on the correspondent WPS and for redirecting its output to the runtime validation endpoint. On the other hand, user tasks, as the name suggests, require user intervention, which allows us to decide, according to the result of the runtime validation, if we want to continue or abort the execution, and “inject” the inputs to the next service task. Obviously, the translation process is implementation dependent, i.e., there is more than one possible translation, per example, we could have used exclusive gateways to perform the functionalities of a user task.

\textsuperscript{20}https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions/spatial

**H. Results**

The results from executing the workflow, figure 11, in Camunda, are represented in figure 16.

Fig. 16: Greece burnt areas in mid July to mid August of 2016, identified and mapped by the workflow of WPSs presented in 11.

The strong red areas are the burnt areas, which correspond to the ones expected in IV-D. Thus, two things were proven:

1) The workflow of WPSs presented in 11 is capable of correctly identify and map burnt areas, given an area of interest;
2) Our solution was successful, since it guided the user through all of the steps needed to complete the workflow, i.e., compose, validate, and translate.

Finally to top it off, the “cherry on top of the cake”, we are, now, able to reuse the same workflow, by saving the template, with a different area of interest, by changing the `source` at the Resample WPS, and execute it without any effort. Hence, creating a pre-defined process to identify and map burnt areas.

**I. Final Considerations**

Altogether, some final considerations must be made, since some aspects might have escaped during this section.

We have found during our investigation in platforms, such as tep\textsuperscript{21} and co-resyf\textsuperscript{22}, that there was, only, one exposed service that performed all of the work. Why does this happen? The WPS standard has grown in a scientific field, where, software engineering practices were not a priority. As a result, WPSs usually are macro-services, this is, encapsulate complex tasks. For this reason, most of the WPSs can not be reused, i.e., there is no notion of micro-services and well-defined responsibilities. These factors and the lack of public WPSs, [17], cripple the advancements in EO, since it does not allow the user to: parallelize operations, share WPSs, understand the execution path due to the tangling of services, and much more.

\textsuperscript{21}https://tep.eo.esa.int/
\textsuperscript{22}http://co-resyf.eu/
V. CONCLUSION

All things considered, we have provided a complete solution to most of the concerns in EO, especially, on the validation of EO workflows, composition, and execution, through translation. Undoubtedly, the approach presented during this document completes the proposed objectives and has focused on contributing to the growth of the geospatial field, by taking into account current studies and researches.

The major achievements and contributions of the present work, concerning EO, are:

- Safe deployment of geospatial workflows to the cloud, hence, avoiding wasting resources - syntactic validation (WPS-V);
- Intuitive modelling of EO workflows - creation automation (WPS-V-WEB);
- Workflow execution in any WMS\textsuperscript{23} - translation;
- An environment, where different skilled users can come together to learn and share their findings, thus, mainstreaming the use of geospatial data and growth of the area.

VI. FUTURE WORK

From my own experience and current researchers in the composition of EO workflows, two things stand out.

Firstly, syntactic validation is fragile and does not give a full picture of the problem, i.e., it lacks the semantic insight of the processes to be able to help the user further. Moreover, our solution should be optimized to deal with the common cases of validation faster, this is, for example, when we want to retrieve the necessary information to validate a bounding box, we have to, first, search for the existence of the CRS and, secondly, query the database for its definition (longitude, latitude, etc). However, the reality is that EO uses a small set of CRSs, hence, we could avoid this problem by, for instance, caching the definitions of the most common CRSs. Additionally, as an optimization we could store intermediate results, thus not losing work already performed and avoiding wasting of resources.

Lastly, as referenced at the end of section III, runtime validation will suffer from network latency, due to the size of the I/O. So, validators should be deployed alongside WPSs to avoid this problem. Likewise, the deployment of workflows of WPSs to the cloud, more than ever, should be a priority, especially given our work. Note that, actually, the future work on the WPS standard will pass through this, since the OGC is exploring an extension to the WPS standard, WPS-T, [15], which will be used to deploy and undeploy WPSs in distributed computation platforms. Equally important will be to understand where and how those services should be deployed to maximize the resources, such as computation speed and money. Surely, models will be studied and developed to cope with these problems.

\textsuperscript{23}It should be capable of using web services and a specific implementation should be done to provide translation to it.

REFERENCES