Workflow Engine for Earth Observation Services

Diogo Rafael Lopes Ferreira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. António Manuel Ferreira Rito da Silva
Eng. Nuno Almeida

Examination Committee

Chairperson: Prof. José Luís Brinquete Borbinha
Supervisor: Prof. António Manuel Ferreira Rito da Silva, Eng. Nuno Almeida
Member of the Committee: Prof. Maria Dulce Pedroso Domingos

October 2018

Dedicated to everyone who helped me along the way, specially, my parents, girlfriend, friends, supervisors, and Deimos for the hospitality and opportunities to present my work.

Acknowledgments

Without a doubt, my parents are the ones that made all of this possible. They sacrificed everything to make me the person I am today. Thus, they deserve the most credit and my honest gratitude. My grandparents, which assured that I had my head on my shoulders, wanted to see me succeed, to grow, reach my goals, and thought me things that I will never forget. My brother, which traced its own path by deviating from the university, inspired me to be positive, ambitious, work hard, and value sacrifice.

Gabriela, my girlfriend, who had the patience to endure this part of my life. She stood by my side from the beginning, as a steady rock, helping me along the way, even without understanding the subject, with English, colours choices, plots, etc. From the bottom of my heart many thanks to you.

My friends both from Lisbon and Barcelos, who I shared terrific times and moments. They helped me to escape from work and to relax.

Of course, my supervisors, Professor António Rito Silva and Nuno Almeida, they were, without a doubt, more than supervisors, they were my friends, on this journey, they taught me more than how to write a thesis, they cared for me, worked as much as me, and gave me knowledge that will become handy in my future career.

Deimos and their employees, which welcomed me as one of them. Thanks for the opportunities to present my work to everyone in the OGC technical meetings and the hospitality.

Finally, IST that made me a competent professional, capable of looking at any problem and with more or less effort come up with a solution.

Resumo

Nos últimos anos, sensores instalados em satélites têm contribuído com enormes quantidades de informação sobre o Planeta Terra. Esta é transformada por cientistas através de ferramentas especializadas, tais como GDAL, SNAP e QGIS, para monitorizar o estado da Terra e prever eventos futuros. Este processo pode requerer mais que uma transformação e gastar grandes quantidades de recursos (CPU e rede).

Dados estes requisitos, com os avanços da internet e o aumento da partilha de recursos, os *web services* tornaram-se numa solução bastante atrativa para os combater. Assim, ferramentas da Observação da Terra que correm localmente, começaram a ser abstraídas como WPSs (Web Processing Services) – um standard especializado desenvolvido pelo Open Geospatial Consortium (OGC).

Por isso, queremos ser capazes de encadear WPSs, isto é, criar um *workflow.* Portanto, surgem três questões fundamentais: como encadear, validar e instanciar *workflows* de WPSs. Genericamente falando, a solução para todos estes problemas pode ser encontrada num Workflow Management System (WMS).

Atualmente, os WMS de outras áreas estão a ser usados para combater as necessidades da Observação Terrestre. No entanto, estes fazem com que se perca alguma da expressividade associada ao standard do WPS, uma peça fundamental para o processo de validação, bem como para a interoperabilidade entre outros OWS (OGC web services).

Assim sendo, e considerando o exposto em cima, propõe-se uma solução orientada à comunidade, por forma a assegurar a interoperação entre os serviços de WPS, através da utilização de JSON para a validação sintática dos *inputs* e *outputs* aquando as fases de modelação e execução do *workflow*, de um modelador para compor e automatizar a sua criação e da possibilidade de tradução da representação interna do *workflow* para a linguagem específica de um WMS.

Palavras-chave: Observação da Terra, WPS, Workflow, Composição, Validação, Tradução.

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.

Keywords: Earth Observation, WPS, Workflow, Composition, Validation, Translation.

Contents

List of Tables

List of Figures

Chapter 1

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.

Given the number of satellites, namely Copernicus[1] sentinel satellites, large volumes of EO data are being generated every day. Usually, these data sets, also called EO products, present themselves as images and their size is on the gigabyte order. Moreover, to be used as analytical data, at least one transformation, i.e., an algorithm, needs to be applied to it. Such algorithms, are present in specialized tools like GDAL[2], SNAP[3], and QGIS[4]. Note that, transformations are, predominantly, resource intensive (bandwidth, CPU, disk space, etc). Additionally, you may require more than one algorithm and product to achieve relevant results.

Thus, analysis over EO data may require many transformations and products, and may scale

---

[1] Copernicus is a European Union Programme aimed at developing European information services based on satellite Earth Observation.

[2] `http://www.gdal.org/`

[3] `http://step.esa.int/main/toolboxes/snap/`

[4] `https://qgis.org/en/site/`

indefinitely, becoming resource expensive. Clearly, having this data available to the public is not enough and technology had to advance to provide a solution to the lack of resources. Remote computation, especially cloud computing, is a natural solution to the previous problem. In particular, Infrastructure as a Service (IaaS) providing infrastructure as shared scientific collaboration platforms across large communities enables data and resource sharing at optimized cost and allows for massively scalable Information and Communications Technology (ICT) infrastructure under pay-per-use models, giving access to resources that users would not be able to afford on their own [2].

Nonetheless, to benefit from the computation in a remote environment, we have to "translate" our tools to the web context. Coming from traditional systems, the way to achieve this is through the use of web services. Web Processing Services (WPS[5]), a standard introduced by the Open Geospatial Consortium (OGC[6]), is a concrete adaptation of web services in the EO field. Moreover, it allows us to "export" these specialized algorithms to the web, by defining a predefined set of operations, inputs and outputs. However, many solutions do not come without additional challenges, and this one is no different. Creating, deploying, and chaining WPSs are now new concerns and require a certain degree of programming knowledge to solve them.

1.1 Problems

In other words, there are large amounts of geospatial data being gathered everyday 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. Having said that, we want to be able to create workflows of WPSs. However, two problems emerge:

- How do we automate[7] the workflow creation, allow its reusability and its deployment to the cloud with confidence[8] 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?

---

[5] `http://www.opengeospatial.org/standards/wps`

[6] `http://www.opengeospatial.org/`

[7] During this document automate means guiding the user, i.e., making its life easier by indirectly giving him capabilities that were done manually.

[8] Confidence here means that the chain represents a valid execution.

1.2 Existing Solution & Solution

Given the problems mentioned above, there is no single solution that solves them in the EO field, as we will discuss in 2. Furthermore, solutions from other areas were adapted without satisfactory results. Nevertheless, lessons were learned, and their concepts can be used to solve these problems.

The solution for the first problem can be achieved through validation of the workflow chain, similarly to what Pegasus does in 2.3.5. A valid workflow chain requires that the connected WPSs are compatible with each other, i.e., one WPS can not follow the previous one, if their inputs/outputs do not match. As a matter of fact, performing validation of WPSs may prove itself difficult, since WPSs are implemented to be used inside a specific community. Moreover, data types, although well defined, may be interpreted in many different ways depending on the implementation. More about these and their full specification is described in 2. So, validating inputs/outputs between web processing services is a requirement to accomplish the chaining of WPSs. In particular, this process of matching data types resembles the responsibility of a compiler when chaining functions, since WPSs can be seen as simple functions. Thus, similar to a compiler, a language will be needed to evaluate the workflow validity, such as the Web Services Business Process Execution Language (BPEL), the Business Process Model and Notation (BPMN), or even a domain specific language (DSL), which only applies to it. As a result, we need to consider which one makes more sense and adapts better to our needs.

Equally important, to solve the second problem there are many solutions, some of them being used commonly on the scientific field such as Taverna, Kepler, and Pegasus while others are used in business processes, such as JBOSS jBPM, Amazon Simple Workflow Service, and Camunda. At the moment, none of these support direct usage of the WPS standard. Furthermore, we should strive to find a solution independent of the workflow engine as stated in [3], which can be achieved by decoupling the engine from the modeler.

1.3 Results

In the end, we were able to solve both problems before runtime, i.e., at the modeling phase by providing a complete solution composed of:

- A validation module, responsible for defining the workflow concept and imposing its constraints, such as data types, number of input occurrences, and input/output validation;
- A workflow modeler, which automates the creation of workflows and templates expressed in a DSL, by not allowing the user to model an invalid process chain, through, per example,

input suggestion and restriction, and by using error messages that come from validating the workflow against the validation module;

- A translation module, which is responsible for translating the internal workflow representation, provided by the modeler, to any target workflow language[9], thus achieving engine independence, and for inserting validating processes between two WPSs to provide runtime validation.

1.4 Thesis Outline

To achieve such solution, we will start by presenting some fundamental domain concepts, i.e., the typical steps in an EO workflow and the WPS standard. Following this, topics from workflow validation, such as the source of data and syntactic validation will be discussed. Then, an analysis is made about current WMSs, both scientific and business, by looking at their adaptation in EO, their functionalities, how they fit the needs of the geospatial area, and the description formats (BPEL, BPMN), used to represent workflows. Having established the background, 2, we will describe our solution in 3, and prove it in 4. Finally, we present some final remarks, including our achievements and future work, in 5.

---

[9] Given that the workflow engine supports web service calls.

Chapter 2

Background

2.1 Fundamental Aspects

2.1.1 EO Workflow Steps

As you might imagine, transforming raw data into usable data requires multiple tools and steps, as it can be seen below in figure 2.1. All the steps will be explained in the following sections in detail.

Data Discovery & Access

The data manipulated by EO workflows is called products. These products, depending on its characteristics, are in the gigabyte order. We can discover these products through, usually, Representational State Transfer (REST) queries to a product catalog. To access it, the user must be authorized, this is, a login has to be made. After authorization, we have basically two options to consult the product catalog: via GUI[1], or through protocols like OData, Open Search, Curl, Wget – which are command line based.

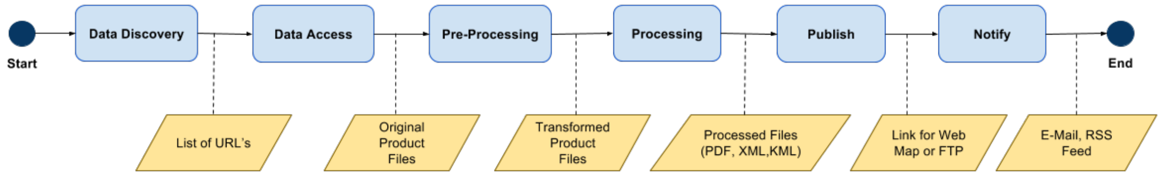


Figure 2.1: Overview of the typical steps in a EO Workflow.

---

[1] `https://scihub.copernicus.eu/dhus`

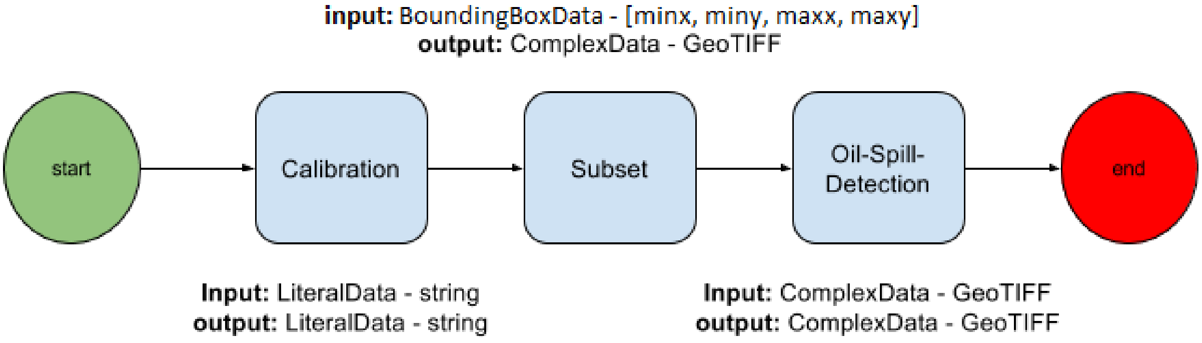

input: BoundingBoxData - [minx, miny, maxx, maxy]
output: ComplexData - GeoTIFF
start
Calibration
Subset
Oil-Spill-Detection
end
Input: LiteralData - string
output: LiteralData - string
Input: ComplexData - GeoTIFF
output: ComplexData - GeoTIFF

Figure 2.2: Overview of a typical EO Workflow.

Pre-Processing & Processing

Products may require some pre-processing before being processed. For example, some processes may require calibration before an oil spill detection process is applied.

To apply transformations and analysis over the products, there are three popular tools in the "market", which are GDAL, SNAP and GRASS GIS. Technically, SNAP abstracts GDAL command line as an interface. As you might imagine, those tools will run in your local computer, but you might want to run them remotely due to resources scarcity, availability, sharing purposes, etc. So, you can also perform data transformations using Web Processing Services (WPSs), which, in essence, create a layer of abstraction on top of these tools, allowing us to call and chain these services through HTTP calls. Therefore, the WPS standard is a key point to distribution, since it exports these tools to a remote context.

Publish & Notify

After all the data transformations are completed, we have to return the output of the Workflow to the user. SNAP provides it via its interface after the operations are complete, GDAL (GPT) demands the user to search through the folders of the resulting product, and WPS exposes it as a response of the REST call.

Workflow Chain Example

This is an example of a chain of WPSs, being the natural representation of a workflow. All the steps mentioned before can be found here. The Data Discovery and Access step can be found, in the chain, on the input of the calibration step, where the Literal Data string is the URL of the product to be retrieved to initialize the chain of processing. Furthermore, since an Oil-Spill-Detection requires Pre-Processing, namely, a calibration, the workflow requires this process.

Additionally, there are two processing steps in this workflow: Subset and Oil-Spill-Detection. Subset allows cropping a product to a particular area of interest based on coordinates or boxes. An example of a Bounding Box Data is a rectangle defined by the tuple (minx, miny, maxx, maxy) - the first two represent the lower corner of the bounding box and the last two the upper corner. Oil-Spill-Detection purpose is self-explanatory, it receives and outputs an image in the GeoTIFF[2] format, which highlights oil spills, if they exist.

2.1.2 Web Processing Service (WPS)

Until now we established that the WPS standard abstracts local tools and algorithms that manipulate EO data remotely, but never presented its specification. 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. Moreover, validating the interaction between two WPS is just a sub-problem of validating a workflow, since the validation of all pairs of WPS, composing the workflow, will result in a valid workflow. 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.

Data Types

WPS[3] defines three types of data inputs and outputs:

1. **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);

2. **ComplexData:** are usually raster or vector files, which can be served as a reference or locally[4]. 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);

3. **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[5]:

---

[2] Is a computer file format for storing raster graphics images with geo coordinates metadata.

[3] `http://docs.opengeospatial.org/is/14-065/14-065.html#25`

[4] Given that they are accessible in the context of the WPS execution.

[5] `http://pywps.readthedocs.io/en/master/wps.html`

- **Identifier:** identifies that data type and must be unique in the process;
- **Title:** usually the same as the identifier, but must be a human-readable title;
- **Abstract:** longer description of input or output, so that the user could get oriented. It allows the user to describe the usage of such input or the output result;
- **minOccurs:** minimum occurrences that the input or output is present;
- **maxOccurs:** maximum occurrences that the input or output is present.

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, and consequently, we will have to dive deeper into them in the next section.

```
<LiteralValue
  dataType=http://www.w3.org/2001/XMLSchema#double
  uom="meter">
    42.1
</LiteralValue>
```

Listing 2.1: Example of an UOM being used.

Operations

WPS provides three important operations, through HTTP Get and Post methods, which will be presented next. But before, it is important to distinguish two concepts used by WPS. The process represents the abstraction of a tool, i.e., an oil spill detection tool, a subset tool, etc. A Service has at least one process. Thus, a WPS may have more than one tool.

Furthermore, processes, similarly to data types, need to define the following attributes: identifier, title, and abstract. They also need to describe the list of inputs and outputs, which should comply with the properties presented in 2.1.2.

A table summarizing the operations, taken from [4], is presented below. There is, also, a section (A) dedicated to representing these operations in more detail, including the responses to the requests.

Operation	Description	Request Example
GetCapabilities	The GetCapabilities operation requests details of the service offering, including service metadata and metadata describing the available processes. The response is an XML document called the capabilities document.	`http://localhost:5000/wps?service=wps&request=getcapabilities`
DescribeProcess	The DescribeProcess operation allows WPS clients to request a full description of one process executed by the service. This description, a XML document, includes the input and output parameters and formats that can be used.	`http://localhost:5000/wps?service=wps&request=describeprocess&identifier=calibration&version=1.0.0`
Execute	The Execute operation allows WPS clients to run a specified process implemented by a server, using the input parameter values provided and returning the output values produced. Inputs can be included directly in the Execute request, or reference web accessible resources.	`http://localhost:5000/wps?service=wps&request=execute&identifier=calibration&version=1.0.0&datainputs=product_url=http%3A%2F%2Flocaldomain%2Fproduct`

Table 2.1: WPS operations table.

2.2 Workflow Validation

A workflow chain has several 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 [5]. Thus, currently very little is done on the validation of workflows.

Nonetheless, before continuing, we should start by clarifying which type of validation we aim to solve. Usually, when we think about validation two types come to mind: syntactic and semantic. The first one focuses on what can be seen, i.e., infers its correctness from what is declared and specified. On the other hand, the second one looks at the real meaning behind, per example, the inputs of a workflow. Naturally, we will shift our attention to the syntactic, since no real validation is performed yet, and it is the stepping stone to achieve semantic validation.

In the light of what was mentioned above, we should verify the workflow in terms of syntactic correctness, before deployment. Syntactic verification serves to know if the model will, at least, run in a Workflow Management System (WMS). Furthermore, validating EO workflows with precision requires both domain and process modelling experts. Domain experts are the scientists and process modelling experts, which in our case will be the developers of the WPS. Ultimately, given the complexity of a workflow, the precision of manual validation[6] decreases with higher complexity workflows. So, it is unrealistic, via manual validation, achieving great results in an acceptable time and precision, hence, analogously, compilers were created for solving this issue in programming languages.

Thus, the end goal would be to automatically check if the workflow is syntactically valid before runtime. In fact, we will go a step further and validate it throughout its execution, and therefore, boosting the user confidence in the work performed. In EO this would translate into running the given EO workflow example, figure 2.2, being sure that, at least, it will not fail due to data validation problems, see table 2.2. Clearly, in tasks that take a long time to perform and consume resources, such as CPU in the cloud, this is a must have, since failing would result in performing the workflow all over again and wasting of resources, as it was introduced in 1. Surely, syntactic validation will add some overhead, since we need to validate the workflow instance before executing it. Note that, naturally, this overhead will increase with the size of the workflow.

2.2.1 Syntactic Validation

Validation is not intended to provide guarantees on the correctness and/or completeness of the workflow. The validation only serves to provide means to check potential problems within the workflow [5]. Thus, validating it does not imply that errors will not occur during runtime. Such example can be seen in the figure 2.2, where it may happen that all the inputs and outputs are satisfied, but during runtime, an error occurs due to mismatching data. More precisely, a process might claim, in its specification (DescribeProcess at 2.1), that its output is of a certain type, but produces a different one.

Generally speaking, a contract should be imposed between two WPSs. Nonetheless, syntactically validation will improve the whole workflow creation process significantly, by weeding out incompatible services.

As a result, the main objectives 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.

---

[6] Manual validation comprises that user that built the workflow has to verify if all its components can be used together.

2.2.2 Data Types & Source

To provide syntactic validation, we need to define a data model. According to [5], there are four types of data: reference, operational, decision, and contextual. Our domain, only, uses the first two. Reference data is usually found on Complex Data types, such as an URL. Operational data comprises all of the other data items required by a process, hence, in EO, Literal Data and Bounding Box Data.

Thus, equally important is the source of data. Not all data required by a workflow is generated by the processes that compose it. Some inputs may be added by the user on the workflow creation. Such example can be seen in the Subset process, where a Bounding Box, defining a rectangle must be given to the process by the user, due to missing data, in order to crop the product into the desired area of interest. Hence, data may not be generated by the links in a workflow. We can then, clearly, distinguish two sources of data: internal and external. Internal data sources are generated within the workflow. External data sources come from the outside environment, usually by reference. Furthermore, external data will have to be typified, this is, since it is not generated by the workflow there is no way of finding the right type of the data, hence when filling missing data, users should be constrained to the inputs supported by the process.

2.2.3 Data Validation Problems

As shown above, two data validation problems were already introduced: mismatching data and missing data. From the possible, basic, validation problems introduced in [5], for the EO domain, we are missing the redundant data problem and insufficient data.

Redundant Data may happen when a process generates more output than what the next process needs. In the figure 2.2, imagine that the Subset Activity generated the crop of the area of interest and the original image. Oil Spill Detection is going to be applied only to the area of interest, which means redundant data was created by the Subset Activity. This may imply two things, from the perspective of the Oil Spill Detection. An activity further down the chain may need this as input, or we can, simply, discard that redundant output. So, input and output matching is not just as simple as seeing if the number of outputs of an activity matches the number of inputs of the next activity. Furthermore, this raises a big question, should we validate outputs that will not be used by processes in a workflow?

Insufficient data is a harder problem to solve. This particular case of data validation comes from the fact that a priori we do not know if even after all data requirements are fulfilled, we can successfully complete the process. Such can happen at a contextual and semantic context, thus depending on the information regarding the underlying applications and expertise of the designers. Insufficient data may also develop from exceptions during a process execution or unavailability of services.

The table 2.2 represents a summary of these problems. All things considered, it would not be too far-fetched that to validate a workflow we would need to express it via some language, so searching for a fitting language, that can express the workflow (data flow, data types, constraints), should be our next concern. Thus, we will see how current WMSs solve this.

Validation Data Problems	**Brief description**	**Example**
Mismatching data	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.	X is required as input for the process B, but A, a previous process, has generated Y different from X.
Missing data	Not enough data was generated by the previous activities to guarantee the execution of an process. Users must express the missing data.	X is required as input for the process B, but there is no source for X, i.e., neither has X been manually introduced by the user or generated by the previous process.
Redundant data	Output data is generated in "excess" from the previous process and is not required in the following process.	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.
Insufficient data	Unavailable services, lack of understanding of the meaning of each data type and its usage, or exceptions running activities.	X is required as input for the process B, but the previous process, A, is not responding due to service unavailability.

Table 2.2: Data validation problems.

2.3 Workflow Management Systems (WMS)

The scientific community is facing, at the moment, challenges related to the increased diversity, complexity, and volumes of data, and the underlying intricacy and heterogeneity of the computing systems that aid the experiments, applications, and data [6].

As a matter of fact, such example can be seen in the executing contexts. Sometimes, executing tasks locally is not enough, because it may happen that you need some particular web service.

In either case, “this complexity and heterogeneity can not be eliminated by the unification of technologies as there are powerful drivers for continued diversity” [6]. Furthermore, we can not “force” a scientific community, which has their own tools and methodologies, to jump into the unknown, abandoning the existing investments to converge into a common technology, which, ultimately, would kill their research momentum. In addition, some legacy systems, usually monolithic, are either too hard to replace, since some sensitive operations require precision and other technologies may introduce errors, or are cost expensive, given that migrating from this architectures into SOA [7] or microservices [8] is a hard task. In either case, it can happen that it is just infeasible. Not to mention that if a common standard were to be introduced, the progress in different fields, due to independent evolution in technologies and requirements, would eventually “break” the standard.

2.3.1 Workflows

A workflow is a list of processes or operations, [9] [5] [6], the set of dependencies between the interconnected processes (the flow), and the set of data resources used to generate or terminate the flow. Generally speaking, workflows are graphs, where processes are the vertices, and dependencies are the edges connecting vertices. These edges can represent control flow or data flow. Additionally, these graphs can be directed cyclic (DCG) or directed acyclic (DAG). More specifically, in EO, usually, graphs are DAGs, since there is no need for the notion of iteration, due to the processing being seen as a sequence of processes (chain), and are data flow graphs, given that the data resulting from processes pass through the edges.

Another important concept is abstract workflow (templates). Abstract workflows define the skeleton of the workflow, as a model or representation. A language is required to represent these to ”translate” the model into a WMS. Then, a WMS will be responsible to: import the template, give the user the opportunity to fill the required inputs/outputs and allocate resources to perform the work.

2.3.2 Workflow Description Formats (Languages)

As we mentioned in 2.2, 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 the workflow. Thus, it is clear that we need to find a fitting language for EO workflows.

From the geospatial domain, there were already studies conducted on the viability of using the Web Services Business Process Execution Language (WS-BPEL), Business Process Model and Notation (BPMN), and a Domain Specific Languages (DSL).

BPEL

BPEL, as introduced in [3], is in version 2.0, and was released by the Organization for the Advancement of Structured Information Standards (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. Moreover, it is perhaps, the most used description format in the geospatial field [10], [11], [12], and [13].

With this in mind, clearly, WPSs would have to be translated into WSDL, in order to use BPEL, which is not far-fetched and actually is a feature of WPS servers, such as PyWPS. However, on the other hand, OGC defines its own protocol stack through OWS, the OGC web services, which is composed by OGC Web Feature Service (WFS), Web Map Service (WMS), Web Map Tile Service (WMTS), Web Coverage Service (WCS) and WPS. Hence, these standards were created to allow interoperability in the domain, so adapting an outside/general standard is a decision that must be pondered. In fact, after many years using BPEL, the following incompatibilities were found, between WPS and WSDL:

- Compatibility of OWS with existing W3C techniques is accomplished by adding further adaptation layers to the protocol stack, making W3C protocols more and more complex [14]. Clearly, using W3C standards improves interoperability, but decreases the conceptual coherence between OWS services, and leaves the advantages of OWS unused. Using BPEL, for example, due to its syntax, it is not intended to be human readable, hence it might be too complex, even for domain experts. Therefore, unlike WSDL, WPS was built assuming human intervention, this is, the interface was tailored for human-to-machine communication [15];

- WSDL does not specify how client applications access WSDL files, because it is not a communication interaction protocol like WPS. This means that each service provider may offer proprietary rules to access the WSDL files. Moreover, there is a need to provision WSDL documents for the corresponding WPS processes, that is, every process inside a service has its own WSDL document, disregarding completely the defined WPS operations (DescribeProcess and GetCapabilities). Besides, an overhead would be introduced by translating WPS to WSDL, [16];

- In the WPS specification, a DescribeProcess request reveals additional process details, such as required inputs and formats, that are essential to validate a workflow syntactically. Due to the generic nature of WSDL, not all the information of a WPS and its processes can be adequately represented. This extra layer of complexity and lack of precision leads to a drastic reduction in the benefits of using WSDL [15]. Hence, there is no one to one match between the WPS and WSDL standards, this is each of the parameters, for example, of the Execute operation must be defined as clearly as possible in the WSDL documents [4]. Furthermore, each input and output is codified using the data type identifier, which has been mentioned in [17] and in 2.3.5, might encounter problems translating the names, and the most similar type will be found to match the WPS and WSDL data types, as seen in [18] and [15].

Thus, it was found that the conceptual mismatches of W3C and OGC approaches are potentially served better finding solutions in other fields, i.e., investigate other description formats. Furthermore, especially, data types incompatibilities are not ideal for validation purposes, since it is a process that needs to be precise.

BPMN

Naturally, as in BPEL, BPMN can not encapsulate the WPS standard, which at the moment is expected, since there is no direct mapping between: the types of data; the processes that compose the workflow (service tasks, user tasks, etc) and WPSs. However, it offers a graphical notation that is easy to understand, by both developers and users. Note that this is a far better approximation to the WPS standard, which has a goal to be human readable. Moreover, the BPMN 2.0 specification is being adopted as an ISO standard, thus gaining a lot of traction in many fields and becoming the language of choice to represent workflows.

Nonetheless, the biggest difference between both languages, is that BPMN integrates into higher technology stacks, such as REST, thus, offering more flexibility. In fact, recent studies, [19] and [20], have tried to use it to orchestrate OGC Web Services.

On the other hand, the BPMN 2.0 specification comes with a lot of unnecessary complexity, i.e., it has many "tools" that are not required, such as, gateways, sub-processes, time events, etc. Even so, it might happen that in the future, with the maturing of EO workflows and the WPS standard, these could be put to use. Regardless, it is common to use subsets of the BPMN specification, thus solving this problem.

DSL

Finally, we can also consider a domain specific language (DSL), thus giving us the freedom to represent the workflow the way we want. It is assumed that a custom, designed language can be optimized for the respective task, since it has a smaller syntax, which can arguably reduce the complexity by dropping none important components that do not apply to the domain. 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 [21]. 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, that ultimately would mean 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.

2.3.3 Workflows & Science

The usual behaviour or approach scientists take on workflows was described in [6] and [22]:

- Scientists compose, operate, analyze, and refine workflows;
- Scientific workflows are exploratory; that is, it is common to reuse workflows and refine them using trial and error;
- Scientific methods are often repeated; that is, scientists rerun workflows with different parameters and datasets;
- Runtime monitoring and diagnostics are important; that is, scientists monitor progress and may steer or decide to abort or suspend execution.

So as the goals:

- Support for collaborative research by enabling scientists to share their findings, i.e., in EO sharing of WPS;
- Abstraction of the details about workflow management and executing, this is, coordinating tasks, allocating resources, etc;
- Integrating resources from distributed and heterogeneous enactment platforms, particularly true in web services;
- Handling large volumes of data and complex computations.

In sum, we should aim to provide ways to reuse and share workflows, abstract its execution from its modelling, be able to handle large amounts of data, and provide monitoring capabilities.

2.3.4 WMS Characterization

We need to create a comparison model to characterize workflows, in order to be able to compare them and find which one, if there is one, fits best to our domain. Naturally, each WMS has its own characteristics, as it can be seen in figure 2.3. Note that every WMS is built to a certain purpose, thus it prioritizes some characteristics over others. As a result, decisions must be made according to these factors. Consequently, to properly classify, discuss, and build a WMS we need to understand these.

Processing elements (PE) are the basic units of computation in a workflow, and they correspond, in EO, to the processes. Executable programs are stand-alone applications that are only accessible in a local context. Web services, on the other hand, can be used remotely, are independent, pre-deployed, and dispersed.

As a result, PEs can be coordinated by two methods. Orchestration, where there is a single controller that oversees the execution flow and invokes services based on workflow chain, and choreography where there is no central entity, thus the responsibility is shared among the participants of the workflow.

In regard to workflow representation, there are many options. A graphical approach will result in a Graphical User Interface (GUI), the most intuitive of them all, but scales poorly in extensive workflows, when there is a lot of processes. Hence, textual representations were created, which, usually, are XML based. The textual representation allows us to express a workflow in a language that can be transformed into an internal representation to be enacted by a workflow engine. Obviously, these types of representations are less intuitive than GUIs. In addition, there are three common internal representations, according to [6], scripting languages, XML-based descriptions, and internal DAG-based representations. These are also used to store the workflow for further changes and reusability.

About the data processing model, it can be divided into bulk or stream data processing. In summary, in bulk data processing, batch processing, we need to receive the whole dataset to perform the transformation versus the stream processing, where we are receiving streams to form a pipeline. Therefore, data in stream processing is processed concurrently, following producer-consumer model [23].

The last characteristic concerns the optimization time. This time has nothing to do with the time that takes to execute a given workflow. Its purpose is to define which phase of the

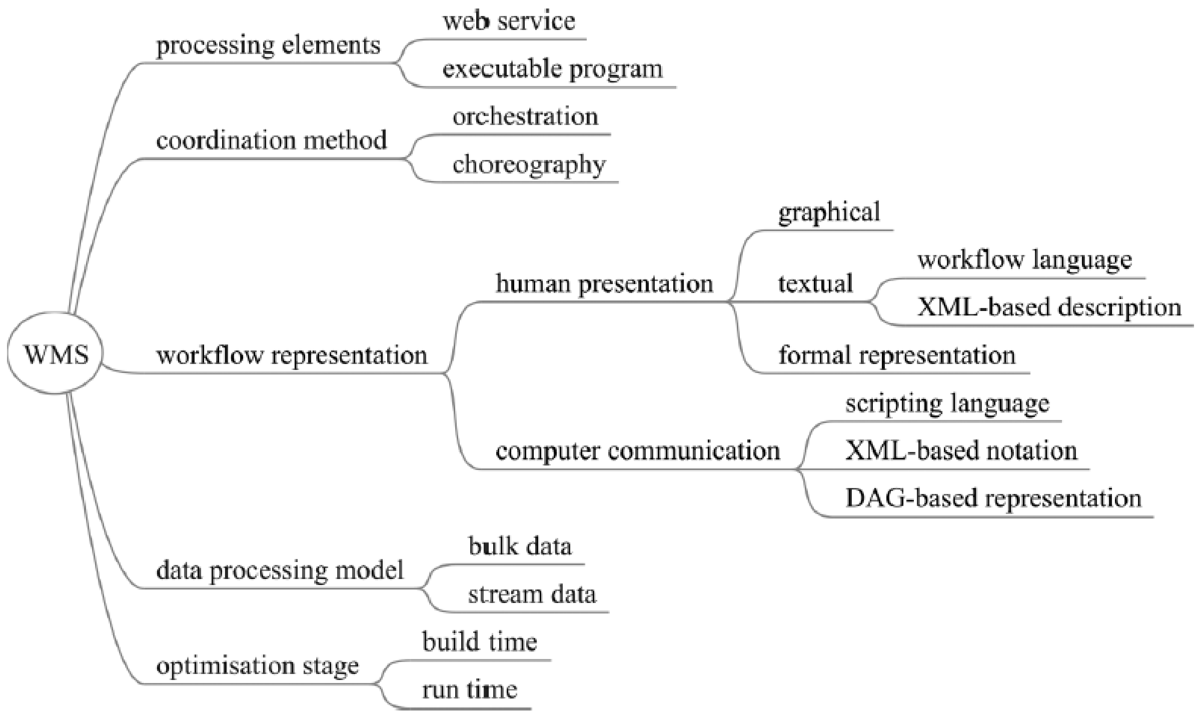


Figure 2.3: Architectural Characterizations of WMSs according to [6].

workflow life cycle the engine will try to optimize the execution. Optimizations can be seen as better scheduling of resources or even workflow validation. In those regards, only workflow validation makes sense, since it is one of the objectives of this thesis. In addition, the scheduling of resources could be a thesis problem by itself, this includes mapping processes to resources, deployment, and connection to DCIs.

Finally, two characteristics that are not contemplated in 2.3 and occur in scientific workflows are monitoring and provenance. The first was already explored in 2.3.3, therefore, the second one, is responsible to record where the data came from, during workflow execution. This is like a log of the data exchanged between activities and its location, in order to be able to reproduce results [24].

2.3.5 Scientific WMS

Below we will introduce three of the most used WMS's in the scientific field according to [6]. Note that none of these were created with EO in mind. EO researchers just saw them as potential candidates to perform EO workflows. Firstly, we will discuss Pegasus[7] and its validator Wings, then we will present Kepler[8], ending with Taverna[9].

---

[7] https://pegasus.isi.edu/

[8] https://kepler-project.org/

[9] https://taverna.incubator.apache.org/

Pegasus

Pegasus is a WMS used in various domains, being one of those EO. It combines Wings, DAGMan and HTCondor, providing, therefore, a complete workflow solution for scientific experiments. Wings is a semantically rich workflow system, used to create/compose, validate workflows and generate metadata. Workflows are stored as workflow templates, which only provide the skeleton of the workflow, without data bindings.

So, Pegasus has the job of mapping the workflow into the execution resources, which includes the data and its location, the computing resources, and the required movements. Thus, Pegasus is a workflow planner, i.e., it does not execute workflows, instead, it delegates these responsibilities to other entities. For this reason, it exploits various engines, for example, HTCondor. Note that Pegasus is capable of exploiting various Distributed Computation Platforms (DCI).

Wings

As described before and in [25], Wings is a semantically rich workflow system. Note that a semantically rich workflow system is not the same as providing semantic validation. Semantically representations of a workflow enables: automated workflow generation; generate metadata attributes for all the new data products of the workflow for provenance[10] and reproducibility; search and discovery of workflows based on their properties; validation of the workflow through the use of ontologies [26]. These features, mainly automated workflow generation and validation, are required in EO.

Validation is made possible due to the fact that Wings uses Web Ontology Language (OWL-DL[11]) as the language to represent the workflow and Jena[12] to validate and reason about it, [27].

The strongest point in Wings is that workflow templates and instances are semantic objects, i.e., all the components, links, data requirements, and data products are represented through metadata with appropriate constraints among them. In fact, metadata is the key to validation in Jena. Moreover, templates ultimately decouple themselves from the creation of workflow instances, which means a scientist can create templates that are widely-accepted analyses and reflect valid scientific methodologies. Thus, less experienced users just have the responsibility of inputting the data. Hence, templates may prove useful in EO, given the referred characteristics.

---

[10] From where the data comes, or was originated from.

[11] `https://www.w3.org/TR/owl-guide/`

[12] `https://jena.apache.org/documentation/inference/`

Kepler

Kepler is a system for reusing, designing, executing, evolving, archiving, and sharing scientific workflows. It became famous for trying to give workflow solutions to many fields like chemistry, geology, molecular biology and oceanography [9].

An attempt to adapt Kepler to use WPS was made on [17], but some challenges were encountered. One of those rises from the fact that Kepler lacks the capability of declaring complex types as the inputs to workflows. If the types are not primitive (int, float, double, string, etc), Kepler assumes it is an object and pushes the responsibility of casting it, to the appropriate type, to the process, this is, inside the code, it will have to typecast. Obviously, this does not translate well in WPS with a vast group of ComplexData shown in table 2.3. Nonetheless, you can circumvent this problem by defining new types of data, but such implies changing the Kepler software distribution. Moreover, parameter types may not be defined declaratively in Kepler, hence there is a mismatch between the Kepler runtime typing and WPS static typing (described through the operations DescribeProcess and GetCapabilities).

WPS datatype	**Mime-type**	**Kepler datatype**
JPEG	image/jpeg	Object
NetCDF	application/x-netcdf	Object
GML	application/gml+xml	Object
O&M	text/xml	Object

Table 2.3: Example Complex Data Type descriptions adapted from [17].

Taverna

Taverna, a domain-independent WMS, was designed to combine distributed web services and/or local tools into complex analysis [28]. It was created to solve problems on composition and enactment of bioinformatics workflows, nonetheless, since it uses web services, it can be applied to other domains.

A particular attempt to marshal the WPS into Taverna was proposed in [29]. In the light of what was shown before, 2.3.2, Taverna uses WSDL descriptions to create processes. So one might conclude, that to use Taverna as WMS, adapted to EO, we need to be able to "translate" the WPS XML into WSDL XML specification. PyWPS was chosen as the tool to bridge the domains, by using Extensible Stylesheet Language Transformation. To cross domains, without considering syntactic validation, incompatibilities must be solved, such as naming conventions and no support for async calls in WSDL. Naming can be resolved by adopting an automated process to transparently map invalid attribute names to valid element names. On another hand, async calls, in WPS, result from "time heavy" computations, that require the user to wait for the results. Since, WSDL was not designed in an era, where such requirement was needed, obviously, some sort of adaptation is required. In their case, they divided an async call into two processes. A detailed description of these challenges and solutions can be found in [29].

2.3.6 Business & Cloud WMS

Similarly to the scientific field, there are solutions in the business processes area to explore. Actually, these are becoming popular in EO and are part of case studies, such engineering reports [3] and testbeds [30]. Thus, in this section, we will explore the most relevant business solutions that were applied to EO. So, we will start by exploring JBOSS jBPM, then we will introduce Amazon Simple Workflow Service, and, lastly, we will talk about Camunda.

JBOSS jBPM

jBPM[13] is a flexible Business Process Management (BPM) Suite. It is light-weight, extensible workflow engine, fully open-source and written in Java. It allows you to model, execute business processes using the latest BPMN 2.0 specification, and monitor business processes and cases throughout their life cycle (authoring, deployment, process management and task lists, and dashboards and reporting).

Thus, with jBPM we are able to model business processes as a workflow of steps, therefore improving the agility and visibility of the business logic. It focuses on executable business processes, i.e., processes that have enough detail to be executed, and on bridging the gap between business users and developers.

In addition to the features mentioned above, this engine provides many other interesting capabilities such as:

- Pluggable persistence and transaction, which is required for provenance as references in 2.3.4;
- Web-based process designer to support the graphical creation and simulation of your business processes (drag and drop);
- Web-based, customizable dashboards for monitoring and reporting;
- Remote API to process engine as a service (REST, JMS, Remote Java API).

---

[13] `https://docs.jboss.org/jbpm/release/7.11.0.Final/jbpm-docs/html_single/`

In sum, it is a complete solution to orchestrate workflows of business processes, since, as we saw above, has many of the desired capabilities in a workflow engine. However its qualities, it was built to the business domain, thus its application to the scientific field, especially EO, is not proven.

Amazon Simple Workflow Service

Amazon SWF[14] helps users to develop workflows, using the Flow Framework, run, and scale background jobs that have parallel or sequential steps. Therefore, it is a workflow engine in the cloud, capable of tracking and coordinating the steps of a workflow. Moreover, it allows the user to track the state of execution, and in case of failure to recover and retry an execution.

Being a product from Amazon it promises, as standard, reliability and scalability. Also, pledges:

- Flexibility, by letting you write your application components and coordination logic in any programming language;
- Logic separation, i.e., promotes a separation between the control flow of your background jobs stepwise logic and the actual units of work that contain your unique business logic;
- Simplicity, since it replaces the complexity of custom-coded workflow solutions and process automation software with a fully managed cloud workflow web service.

The deployment of EO workflows on Cloud environments was already investigated by previous OGC Testbeds and in research, [31], [32]. Nevertheless, their goal was to compose workflows of web services deployed in the cloud, not the use of Amazon SWF itself for the composition.

Camunda

Camunda is a Java-based framework supporting BPMN 2.0 for workflow and process automation. Their main objective, as in 2.3.6, is to shorten the gap between business users and developers, by adopting BPMN you can express reliable service orchestration, human task flows, event handling and much more in diagrams that are technically executable yet easy to understand for everyone.

As an engine, it prides itself on being simple to use, and by exploiting the most recent technologies and standards. In fact, its simplicity comes from the vast range of services (Camunda Stack) that it offers, such as workflow modelling, task lists, monitoring (cockpit), administrator panel, and optimization services.

---

[14] `https://aws.amazon.com/swf/`

It can be integrated with any maven project, spring boot, or even Java EE. Moreover, Camunda can be used externally through rest calls, i.e., standalone, thus a user can access the Workflow Engine via REST in order to start process instances, complete tasks and much more. Hence, we can develop and operate our (micro-)services completely decoupled from the Workflow Engine and let them pull the work via REST whenever it suits us.

Equally important, Camunda offers detailed documentation, video tutorials, an active community, complete examples, and open source projects. As a result, Camunda is gaining traction in many fields, especially in EO [3], [30], because it exploits BPMN 2.0 and is easy to install, which is required for producing testbeds or engineering reports[15]

2.3.7 Comparing WMS's

All things considered, the summarized version of the characterization of the WMS presented above can be found in table 2.4. Note that, every WMS has the same coordination method (orchestration), data processing model (support to bulk data, which is what matters in EO), and some sort of support for provenance. Furthermore, validation, here, corresponds to a dedicated validation module, not language validation, as we have in BPMN and BPEL.

	Pegasus	**Kepler**	**Taverna**	**jBPM**	**SWF**	**Camunda**
Processing Element	executable program	both	both	both	both	both
Representation	textual	graphical	both	both	both	both
Validation	yes	no	no	no	no	no
Language	dax	ptolemy's	t2flow	bpmn 2.0	flow framework	bpmn 2.0

Table 2.4: Summarized version of the characteristics of the WMSs presented.

Obviously, in the light of what was shown above, there is no solution that will fulfill the demands of every field and if there were it would eventually diverge. In the next section, we will discuss our solution to the EO field with WPS and systems presented in mind.

---

[15] Testbeds or ERs are conducted in the EO domain as a prototype to test the waters to future changes to the standard.

Chapter 3

Implementation

In order to explain our solution, i.e., our implementation, we should revisit the problems introduced in 1.1, that we were trying to solve:

- 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?

During the chapter 2, we have introduced all of the fundamental concepts and different approaches to deal with these. Thus, we will start by reflecting on the core modules and their responsibilities, then we will see how WPS-V[2] module validates EO workflows, after that we will present the WPS-V-WEB[3] module and its features, and, lastly, we explain how we validate workflows at runtime.

3.1 Modules

We have two core modules: the validation module (WPS-V) and the web module (WPS-V-WEB). Their separation and creation were motivated by the requisites presented along this

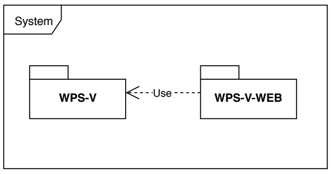


Figure 3.1: System overview.

---

[1] Confidence here means that the chain represents a valid execution.

[2] `https://github.com/drlferreira/WPSV`

[3] `https://github.com/drlferreira/WPS-V-WEB`

document. It should be mentioned, as it can be seen in the figure 3.1, the web module uses the validation module, in order to provide its functionalities to the user. Note, however, that the validation core can be used by any other project, being independent. Lets now separate the problem solving between the modules:

- **WPS-V:** is responsible for providing validation of EO workflows, which allows the user to deploy them with confidence to remote infrastructures that may use a pay-per-use model or resource scheduling plans.
- **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; and lets the user execute the workflow. Therefore, the web module, clearly, solves the automation, reusability, and execution problems.

3.2 WPS-V

With the rise of WPS-T, [3], transactional, a validation module makes more sense than ever, since WPSs are being sent to the cloud to acquire proximity to the data and a workflow that does not run will spend/allocate resources without optimal use.

Therefore, the pragmatic behind the creation of this module is to provide a safe deployment to the cloud, avoiding waste of resources (money or scheduling time). Moreover, it is used to support the web module, providing guidance to the user creating the workflow. Such can be obtained by syntactically validating the workflow chain. Clearly, if the chain is valid then we can perform its deployment. Nonetheless, it should be mentioned that this validation occurs, first, in the build time and, secondly, in runtime.

As we saw in 2.2, to provide syntactic validation on EO workflows, we need to take into account the description formats (BPMN, BPEL, DSL), 2.3.2, the data types and their source (internal, external), 2.2.2, the phase of validation (build time and runtime), 2.2.1, and lastly the data validation problems (mismatching, missing, redundant, insufficient), 2.2.3.

3.2.1 Syntactic Validation: Description Formats

As we introduced in 2.3.2, there are many languages to represent a workflow, being our options BPEL, BPMN, and a custom DSL. We have discussed in 2.3.2, that BPEL was no fit for encapsulating the WPS standard, neither BPMN as we saw along 2.3.2. Our conclusion was that the WPS standard, and the OGC stack, was too specialized to be adopted by any already existing language for validation purposes. 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.

3.2.2 Syntactic Validation: Build Time and Runtime

This pragmatic motivated 3.2.3, since usually, in compilers or even in 2.3.5, syntactic validation is provided at the building phase. In workflows, the building phase corresponds to the chain composition. Clearly, at this stage, we 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. Thus, we provide syntactic validation at both phases.

3.2.3 Syntactic Validation: Validation problems

We distinguish 4 levels to accomplish syntactic validation in geospatial workflows, which can be seen in figure 3.2. 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 need to be sure that the level 0 is satisfied. Being said that, now, we will ”climb the staircase”.

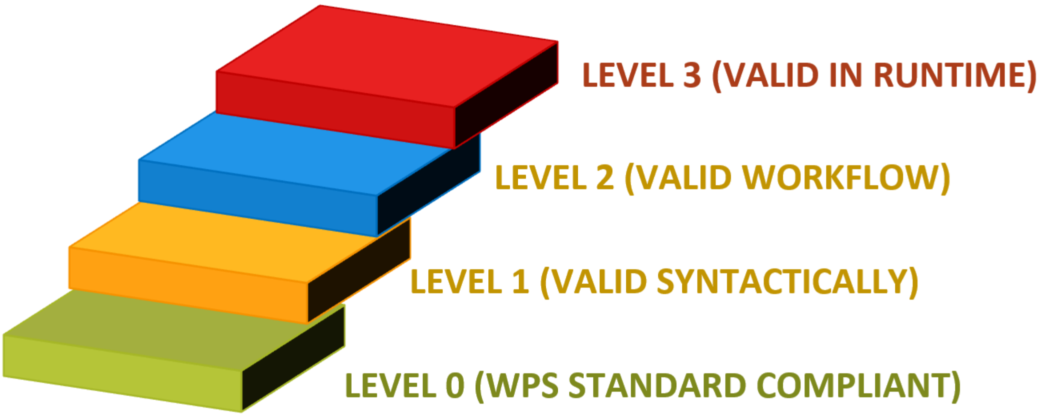

LEVEL 3 (VALID IN RUNTIME)
LEVEL 2 (VALID WORKFLOW)
LEVEL 1 (VALID SYNTACTICALLY)
LEVEL 0 (WPS STANDARD COMPLIANT)

Figure 3.2: Levels of syntactic validation in EO workflows.

Level 0

Level 0 is the most trivial validation of them all. It checks if the Describe Process of a WPS process specification (xml) obeys the schema (xsd), see figure 3.3. Note that, it would not make sense trying to validate something that does not obey the standard.

Fulfilling this level, allows us to parse the WPS process to an internal representation, through JAXB[4], that can be used in the validation process. This internal representation will then be used as a foundation for the validation.

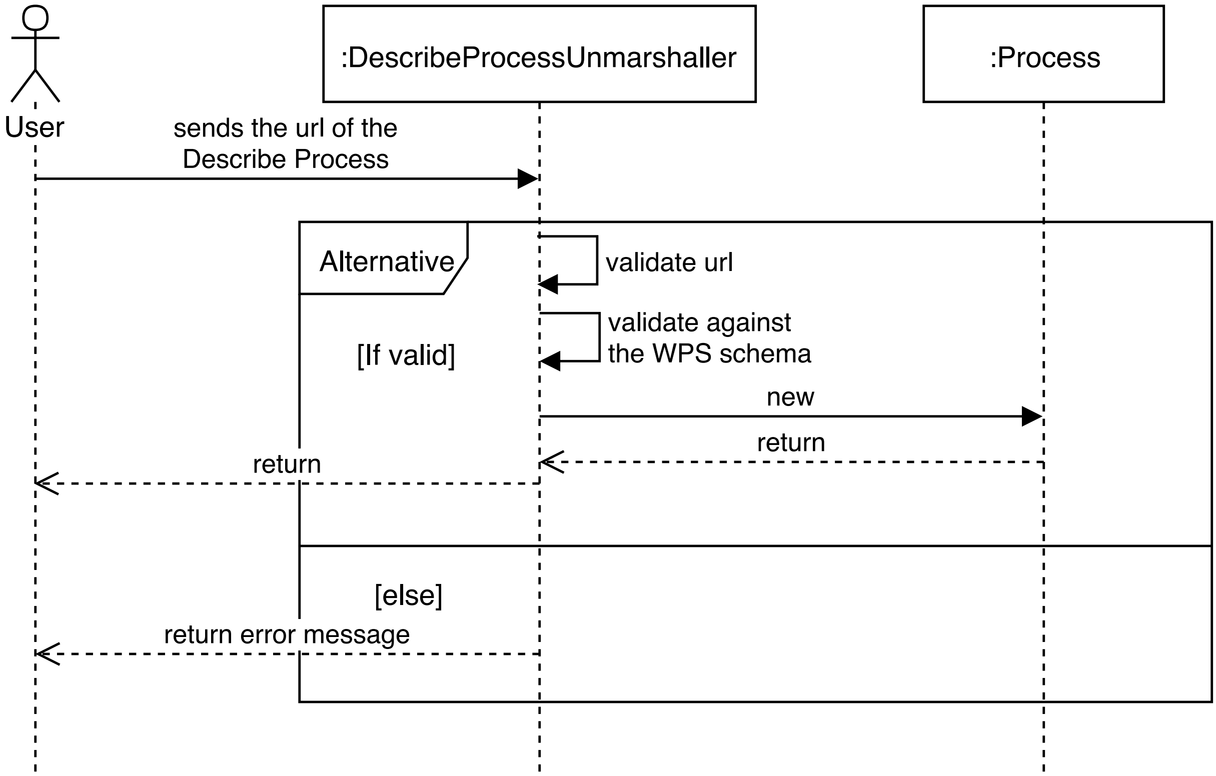

:DescribeProcessUnmarshaller
:Process
User
sends the url of the Describe Process
Alternative
validate url
validate against the WPS schema
[If valid]
new
return
return
[else]
return error message

Figure 3.3: Sequence diagram representing the interactions in Level 0.

Level 1

Now that we have an internal representation of the WPS processes, we are ready to, given an input, validate it against its specification, and hence solving the mismatching of data. Thus, this level assures that all the data types and their restrictions are respected. As a result, our next task is to understand what are the validation problems, for each of the types of data.

- **LiteralData**
  - *Data Type*: validate the input against its specified data type. They can be a string, integer, decimal (float and double), boolean, datetime, date, and time;

---

[4] JAXB comes as standard in Java 8 and allows parsing of XML to domain classes.

  - *Range*: check if the value is within the range. Ranges can go from $]-\infty, +\infty[$ and may be Open, OpenClosed, Closed, ClosedOpen. These are used to let us know if the first or last values should be included[5]. For OpenClosed an example would be $]-4, 56]$. In addition, it may define a spacing, which defines the range step. Considering the example mentioned above, if we have a spacing of two, the allowed values would be -2, 0, 2, 4, and so on;
  - *Allowed Values*: confirm that the input is within this list of allowed values.

- **BoundingBox**
  - *CRS (Coordinate Reference System)*: check if the values are within the bounds of the chosen CRS. What distinguishes this coordinate reference system from the others is that, in this one, its axis are defined by longitude and latitude. There are different types of CRSs supported such as: EPSG:4326, WGS 84, and NAD83.

- **ComplexData**
  - *MimeType*: validate if the input obeys the specified mime type. These are: application/json, application/gml+xml, image/tiff, application/x-zipped-shp, application/x-ogc-wms, application/x-ogc-wcs, application/x-ogc-wfs, etc...
  - *Encodings*: confirm that the input is written in the specified encoding, which can be, per example, utf-8, utf-16, raw, and base64;
  - *Schema*: if the mime type is application/json or application/xml, then we verify against their schemas;
  - *MaximumFileSize*: the file size must be $\leq$ that maximum file size acceptable.

In the light of what was shown above, 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 3.4, which was inspired by [33], where they used the XML constraints from the specification to build reusable blocks of validation. These are described below.

From figure 3.4, 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 homogeneously procedure, where at the process creation, we attach a Process Validator that will delegate the validation to the correct validator.

---

[5] The same concept as intervals in mathematics.

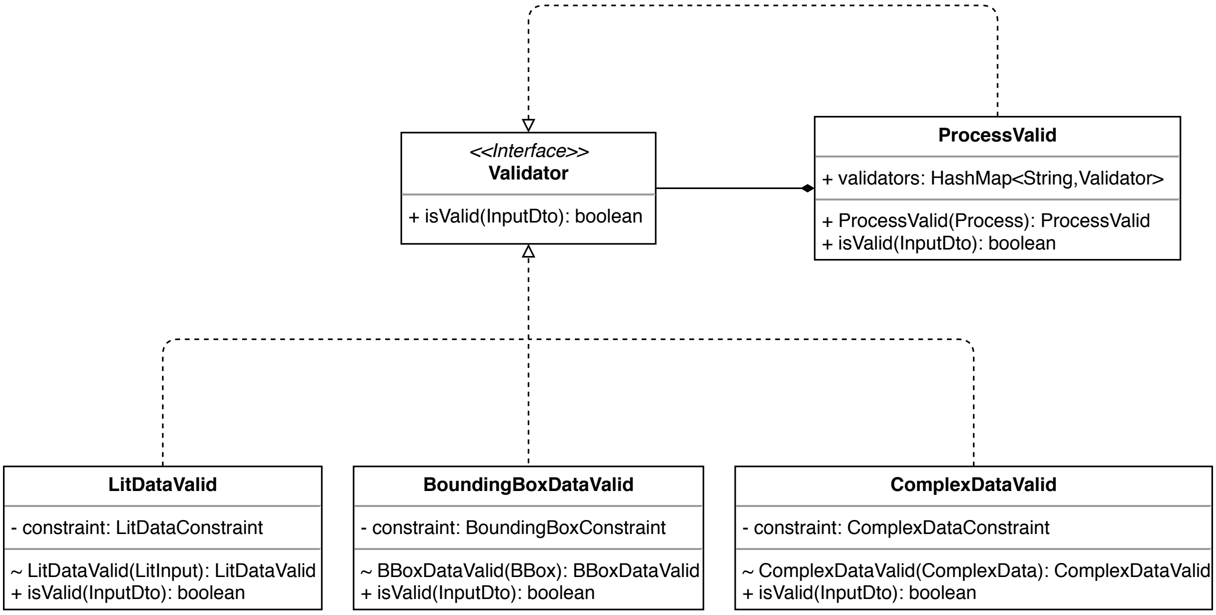

<<Interface>>
Validator
+ isValid(InputDto): boolean
ProcessValid
+ validators: HashMap<String,Validator>
+ ProcessValid(Process): ProcessValid
+ isValid(InputDto): boolean
LitDataValid
- constraint: LitDataConstraint
~ LitDataValid(LitInput): LitDataValid
+ isValid(InputDto): boolean
BoundingBoxDataValid
- constraint: BoundingBoxConstraint
~ BBoxDataValid(BBox): BBoxDataValid
+ isValid(InputDto): boolean
ComplexDataValid
- constraint: ComplexDataConstraint
~ ComplexDataValid(ComplexData): ComplexDataValid
+ isValid(InputDto): boolean

Figure 3.4: The structure of the different types of validators for Level 1.

Regarding the constraints, it should be noted that every constraint 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 *BoundingBoxConstraint* 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 3.5.

A *BoundingBox* has a list of supported CRSs, 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[6], 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.

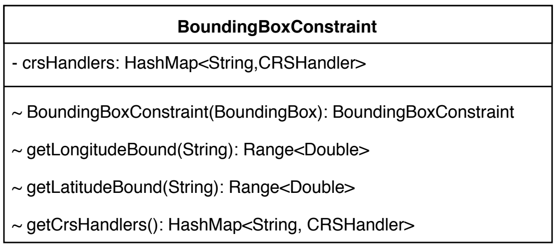

BoundingBoxConstraint
- crsHandlers: HashMap<String,CRSHandler>
~ BoundingBoxConstraint(BoundingBox): BoundingBoxConstraint
~ getLongitudeBound(String): Range<Double>
~ getLatitudeBound(String): Range<Double>
~ getCrsHandlers(): HashMap<String, CRSHandler>

Figure 3.5: A *BoundingBoxConstraint* has into account the validation problems mentioned above as we can 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.

---

[6] `http://sis.apache.org/`

Algorithm 1: BoundingBoxData Validator pseudocode.

```
$input \leftarrow$ BoundingBoxDto;
if $crs \in getCrsHandlers$ then
    if $input.longitudeBound \in getLongitudeBound \land input.latitudeBound \in getLatitudeBound$ then
        return $true$;
    else
        throw new ValidationException;
    end if
else
    throw new ValidationException;
end if
```

It is easily extrapolated, based on the validation problems, how we could construct the other constraints. As a clue, *ComplexDataConstraints* support themselves on Apache Tika[7], 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 3.2.3. Moreover, *LiteralDataConstraints* use Google Guava[8] to define *Ranges*, and Java *XML DataTypeConverter* to verify *Data Types* against 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 it was 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.

Level 2

The previous step gave us valid links in the workflow chain. Remember if every process that composes the chain is valid then we have a valid workflow. As a result, in level 2, we will look at the workflow specification, written in JSON, verify it against the schema and import 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:

- *The internal inputs compatibility*;
- *Minimum & Maximum Occurrences*: it is defined in each process input how many occurrences it supports, the minimum and maximum. Note that a process may be optional if the min threshold is set to zero.

---

[7] `https://tika.apache.org/`

[8] `https://github.com/google/guava`

The solution, to level 2, can be seen at figure 3.6. 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[9]. Lastly, we will send the workflow object to *WFValidator*, which was constructed based on figure 3.4.

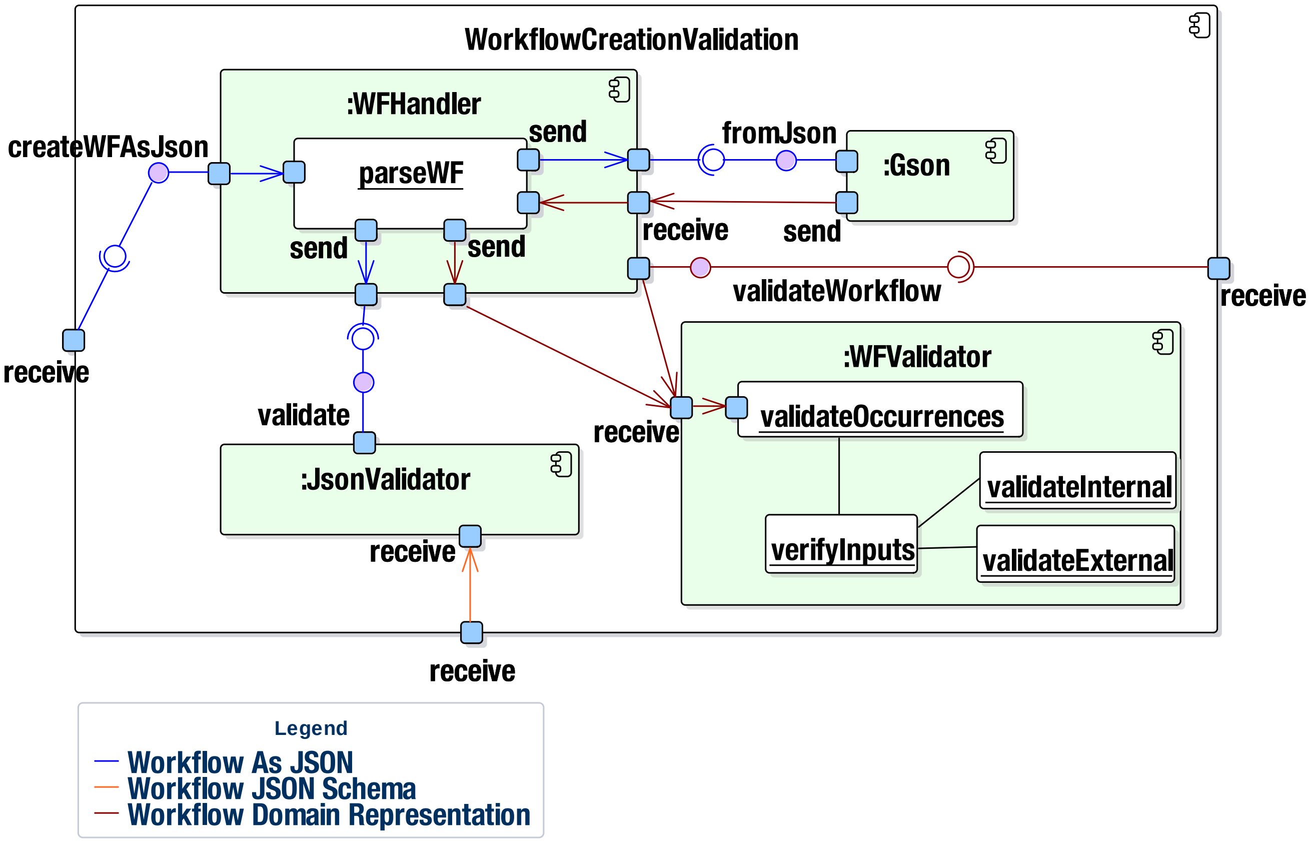


Figure 3.6: Level 2 Component and Connector diagram explaining the runtime behaviour when creating and validating an workflow.

At this stage, we are ready to validate the workflow with the following objectives in mind:

- *validateOccurrences*: check if the number occurrences are satisfied (missing data), by looking at each WPS process specification and comparing it to the number of inputs supplied;

---

[9] `https://github.com/google/gson`

- *verifyInputs*: skips the redundant inputs, i.e., that will not be used in the workflow chain. Its implementation is naturally provided by the iteration and recursion of the chain, since we do not explore inputs that are not used as dependencies to other processes. In addition, at this point, we look for insufficient data, after all, we are testing the submitted inputs, for example, in a ComplexData, we test if the URL, for a product, is available and valid;
- *validateInternal*: validate the internal inputs compatibility. An output of a process is considered compatible with the input of the next if they share commonalities in their specification. For instance, in a LiteralData, it is required being of the same type of data, having the same data type, and the output unity of measure needs to be supported by the receiving process;
- *validateExternal*: is provided by the previous level, thus already explained in 3.2.3.

Level 3

In this last step, motived by 3.2.2, we already have a valid workflow, so its job is to make sure that in runtime it stays in that way, i.e., we need to verify the outputs that are being generated within the workflow and are being used by other processes. Note that its implementation will be left to the end of this chapter, where all the context has been provided.

3.3 WPS-V-WEB

This module was created to fulfil the requirements and functionalities presented in 2.3.3. We can easily map those into the subsections of this section. In sum, WPS-V-WEB is a blackbox that abstracts the complexity of managing a workflow, by exploiting the WPS-V module, 3.2. The technology stack is composed of: Spring Boot[10] for the backend, and Vue.js[11] for the frontend supported by Buefy[12] (combines Bulma[13] into Vue.js components).

At this point, it should come as no surprise, after an extensive description of the different Workflow Management Systems, 2.3, that they are the answer for our problems, since they have all the requisites necessary. Thus, it would be naive not to base ourselves on them. However, it should be regarded that, at the moment, we already defined our own DSL due to our validation requirement, thus we are forced to discard their usage, which implies the creation of a specialized EO WMS. Yet, engineering taught us to reuse previous work. So, with that in mind, below, we present our solution.

---

[10] `https://spring.io/projects/spring-boot`

[11] `https://vuejs.org/`

[12] `https://buefy.github.io`

[13] `https://bulma.io/`

3.3.1 Workflow Creation Automation

We have characterized WMSs in 2.3.4, and analysed many of them, from both scientific and business fields at 2.3. Without a doubt, one of the trending features, as it can be seen in table 2.4, was the possibility to compose workflows via a graphical interface. GUIs, surely, help to intuitively build workflows. Thus, we have built a canvas, figure 3.7, based on diagram-js[14], where the user can drag-and-drop WPS processes and chain them together. Its palette is based on a extremely small subset of the BPMN graphical specification, only providing the user start events, service tasks, and end events.

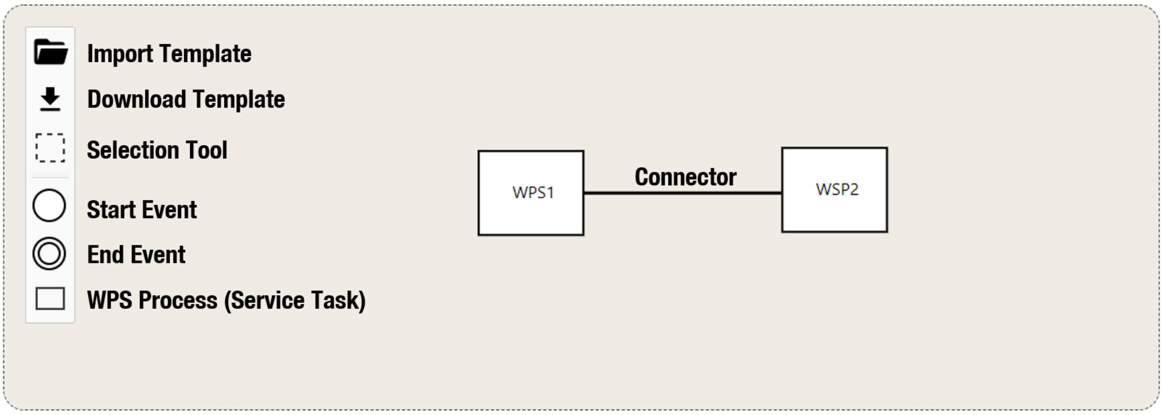
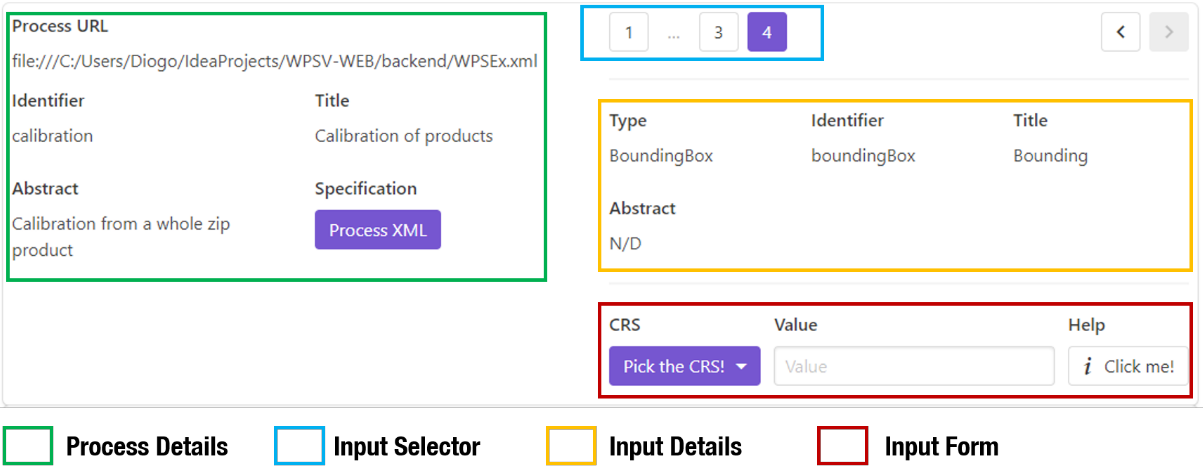


Figure 3.7: WPS-V-WEB diagram canvas.

In addition, this web module helps the user fill missing inputs, by showing only the relevant information to satisfy it. This is supplied, by the process component, figure 3.8



Process URL
file:///C:/Users/Diogo/IdeaProjects/WPSV-WEB/backend/WPSEx.xml

Identifier: calibration
Title: Calibration of products
Abstract: Calibration from a whole zip product
Specification: Process XML

Type	Identifier	Title
BoundingBox	boundingBox	Bounding

Abstract
N/D

CRS	Value	Help
Pick the CRS! ▾	Value	*i* Click me!

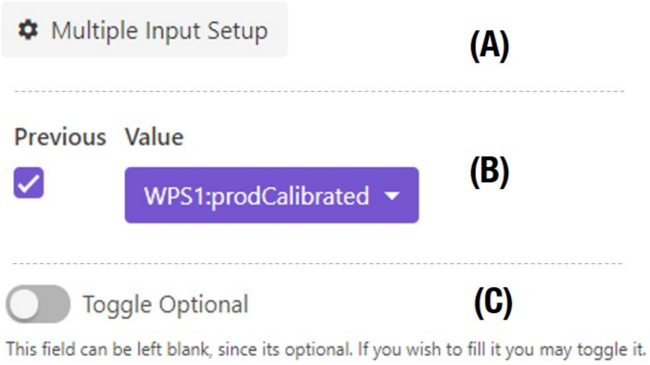
Figure 3.8: Process Component in WPS-V-WEB, where we can see how the user is helped in a BoundingBox external input.

---

[14] `https://github.com/bpmn-io/diagram-js`

The process component is, therefore, composed of four ”areas”. Both *Process Details* and *Input Details* are self explanatory. There is only one particularity, at the Process Details, which is that we provide the ability to view the WPS Describe Process specification, because it may happen that a domain expert requires to inspect it for further context information. Likewise, the *Input Selector*, naturally, as suggested by its name, serves to select the input of a WPS Process. Finally, the *Input Form* is where the magic happens, since it constraints and provides guidance to the user when filling the input. For instance, the selected input, a BoundingBox, at the *CRS Picker* informs the user that the supported CRS is EPSG:4326, thus, when fulfilling the value, the user should take it into account. Moreover, it also provides a helper to check how the value should be formed.

Besides all of these features, it goes a step further, the *Input Form* adapts itself, figure 3.9, according to: multiple input setup **(A)**, i.e., the possibility to reuse an output from a previous process **(B)**, and an input being optional **(C)**.

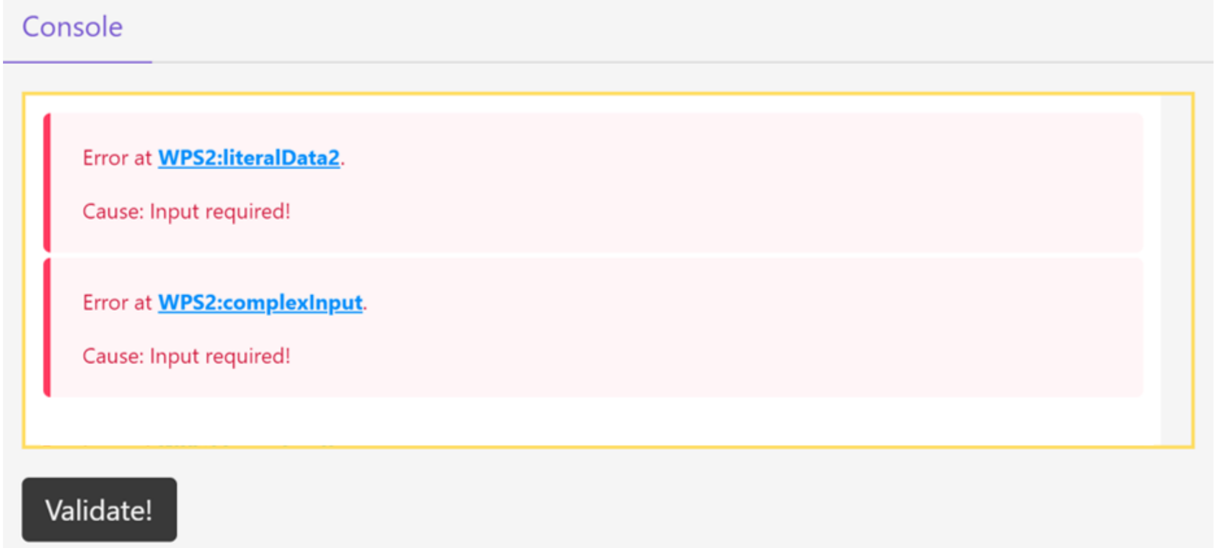


This field can be left blank, since its optional. If you wish to fill it you may toggle it.

Figure 3.9: The different Input Forms in the Process Component.

Regarding the multiple inputs **(A)**, 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, 3.2. These are presented in a console, similarly, to compilers, figure 3.10. 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.



Error at WPS2:literalData2.

Cause: Input required!

Error at WPS2:complexInput.

Cause: Input required!

Figure 3.10: WPS-V-WEB console, which display error messages, resulting from validating the workflow.

In conclusion, it is undeniable that this tailored frontend provides a composition environment that nourishes the workflow creation automation.

3.3.2 Workflow Reusability

The solution to workflow reusability was first introduced in 2.3.5, where it was possible to download the workflow template, without the data bindings, to be reused for further experiments. Remember that this a major requisite in workflows, 2.3.4, since they are common to be reused, refined using trial and error, and rerun with different parameters and datasets.

Thus, it goes without saying that, the answer to this problem is 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, 2.3.2, on a text file, listing 3.1, that can be downloaded by clicking in the *Download Template*, figure 3.7. Obviously, the reverse can also be done, i.e., imported.

```
{
  "workflow": {
    "identifier": "710894ce-f216-4aaa-ace7-c35e46c15f77",
    "owner": "6dee8ac0-c508-11e8-92de-f512710b39b4",
    "chain": [
    {
      "index": 0,
      "task": {
        "processURL": "WPSV-WEB/WPSEx.xml",
        "schema": "http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd",
        "inputs": [],
        "outputs": []
      }
    }
    (.............................................)
    ]
  },
  ”diagram”: [
  {
    ”width”: 100,
    ”height”: 80,
    ”url”: ”WPSV−WEB/WPSEx.xml”,
    ”name”: ”WPS1”,
    ”id”: ”shape_13”,
    ”x”: 363,
    ”y”: 154
  },
  (.............................................)
  {
    ”id”: ”connection_19”,
    ”waypoints”: [
    {
      ”x”: 767,
      ”y”: 197
    },
    {
      ”x”: 910,
      ”y”: 193
    }
    ],
    ”sourceId”: 1,
    ”targetId”: 3
  }
  (.............................................)
}
```

Listing 3.1: WPS-V-WEB Workflow Template.

3.3.3 Workflow Execution

Our workflow definition is now valid, since we have taken our workflow JSON definition and used the WPS-V module to validate it, thus, we are finally ready to execute it. However, if you remember, 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. Imagine implementing all of the requirements presented in 2.3.4, and concretized/refined in 2.3.5 and 2.3.6, surely, it would be way out of this thesis scope.

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 obvious, 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.

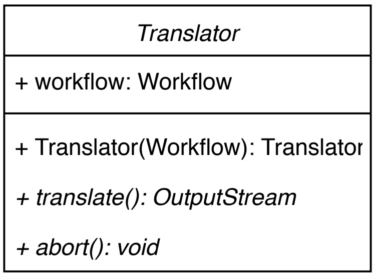

Translator
+ workflow: Workflow
+ Translator(Workflow): Translator
+ translate(): OutputStream
+ abort(): void

Figure 3.11: WPS-V-WEB translator interface.

So, if we want to use, per example, jBPM to manage our workflow execution, we would need to translate our internal workflow representation to BPMN 2.0. Hence, we have provided a *Translator* interface, figure 3.11, that can easily be extended to provide such functionality. In this particular case, we would create a specialized translator for BPMN, which would from the *workflow* construct its BPMN specification. Furthermore, being BPMN part of the ISO standard, we now would be able, without any effort, to execute the same workflow in Camunda.

3.4 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, as seen in 2.3.4, executable processes or web services. Therefore, we have added processing elements that call our application through a rest endpoint, listing 3.2, 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 3.2: 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 PEs, 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[15].

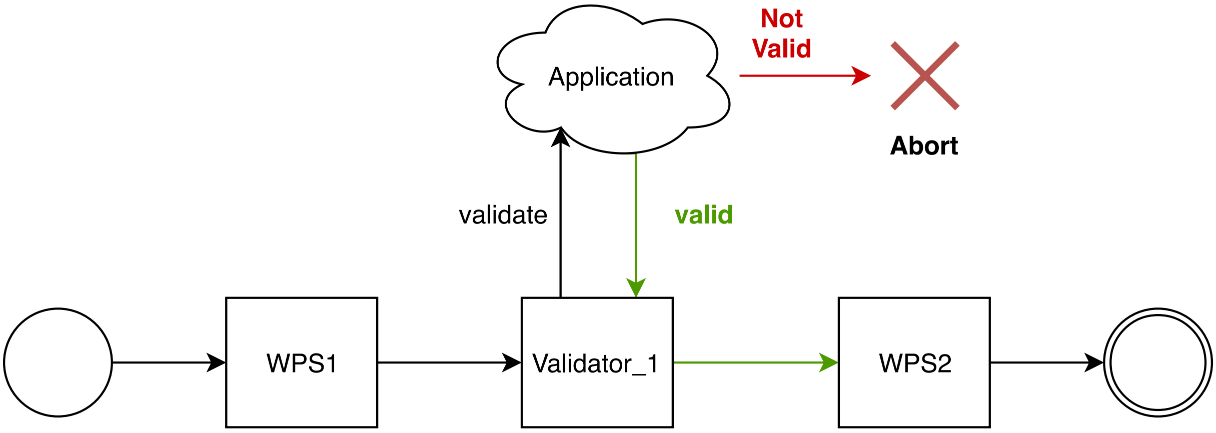


Figure 3.12: Schematic for our solution to runtime validation.

In sum, it can be seen, in figure 3.12, 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 3.11, which can be used to, per example, abort the execution gracefully, i.e.,2 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.

3.4.1 Costs

As expected, our solution to validate the workflow at runtime has associated costs. These affect, primarily, the performance, due to the increase on the number of communication events. In fact, validation, by itself, introduces overhead, especially, in schema validations and mime types (ComplexData). Consequently, the overhead grows with the number of links in the workflow, however, its growth is not proportional, since it depends on the nature of the workflow, this is, the elements that compose it.

From figure 3.12 we can conclude that for each link in the workflow, we require one validator process, except for the first and last links. Nonetheless, according to the user translation process, this might not be true, since the user might want to validate the final output of the chain, as it will be seen in the next chapter.

---

[15] The process to parse them is just as 3.2.3.

So, if there are L links, there will be $L-1$ validators and, thus, the number of communication events $(L-1)+c$ will become $(L-1)+(L-1)*2+c$, where $c=2$ (the communication from the start event to the first link, and from the last link to the end event).

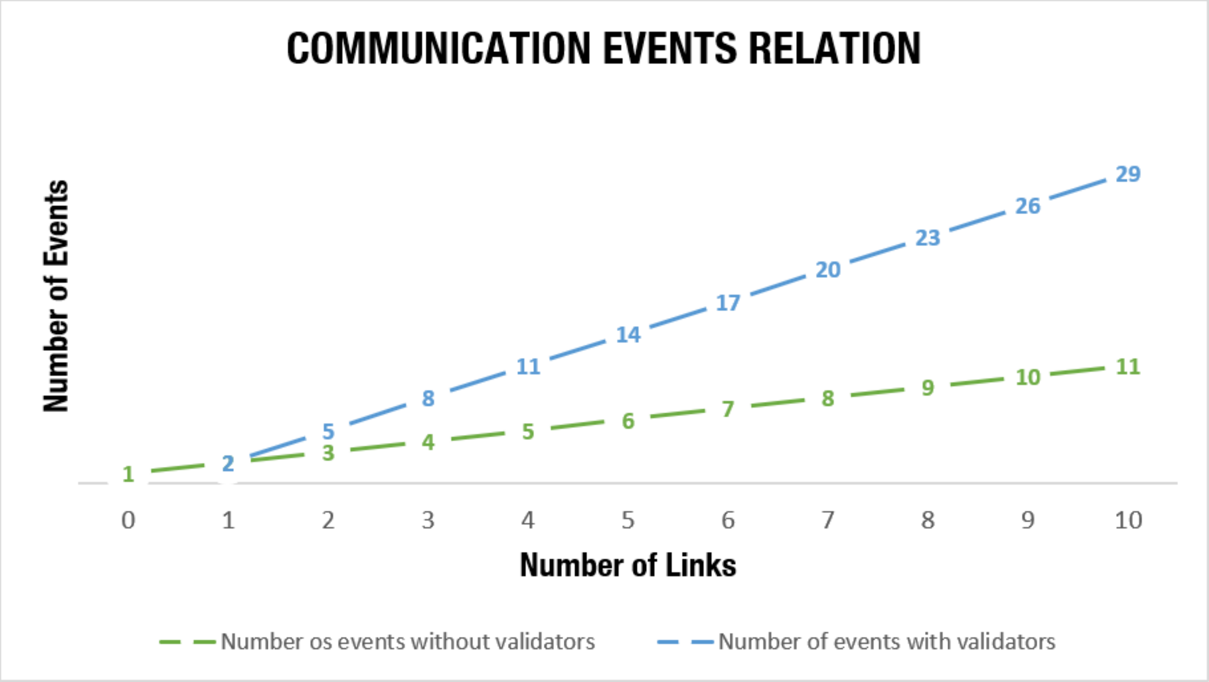


Figure 3.13: The increase on the number of communication events by introducing validators in the workflow to validate it in runtime.

Therefore, looking at the graph and expressions above, we can conclude that for each link added, we increase $2*L\ ;if\ \ L>1$ communications events. Usually, these can correspond, if the links are in different machines, to network latency, that will vary according to speeds, location of the processes, size of the packets, and other factors. As a result, we can not quantify this in units of time. However, there is a big catch, i.e., we are forcing the outputs of a process to do extra hops to be validated. Ultimately, this, in EO and 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.

As a final remark, it should be noted that the overhead introduced by the validation itself, i.e., having an input and checking its validity internally disregarding if it is build or runtime, is marginal in relation to the time that most geospatial algorithms, encapsulated in the WPSs, take to execute. Consequently, it is more than worth to validate the workflow at the build phase, because the time we would lose by an invalid execution would be much greater. With all of these factors in mind, runtime validation, to be effective, should be supported by orchestration techniques that take into account data movements. All the same, this topic is out of the scope of this thesis.

Chapter 4

Results

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[1].

4.1 Sentinel-2 Imagery to Identify and Map Wildfire Events

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 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.

---

[1] `https://www.mallontechnology.com/`

4.1.1 Benefits

- More accurate post-event delineation of wildfire extents;
- Smaller areas affected by wildfires can be determined and assessed;
- The need for ground surveys on often difficult terrain to determine wildfire locations and extents can be potentially reduced, or made easier;
- Additional data can be used in conjunction with the imagery by a variety of organisations to better assess, monitor and respond to the areas and communities affected by such wildfires;
- The information obtained can be used to carry out risk-assessments of affected and surrounding areas;
- Sentinel-2 data collected over time can be used to monitor environmental recovery, especially in relation to agriculture and forestry.

4.1.2 Process Steps

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

1. Access the Sentinel Scihub[2] (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).

It should be noted 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 work efficiently, i.e., cut the processing time. Furthermore, all of these steps require a lot of

---

[2] `https://scihub.copernicus.eu/dhus/`

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 4.2.

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[3] to identify and eliminate the clouds.

4.1.3 Workflow Schematic

We can build a workflow using the steps found in 4.1.2, which can be seen in figure 4.1.

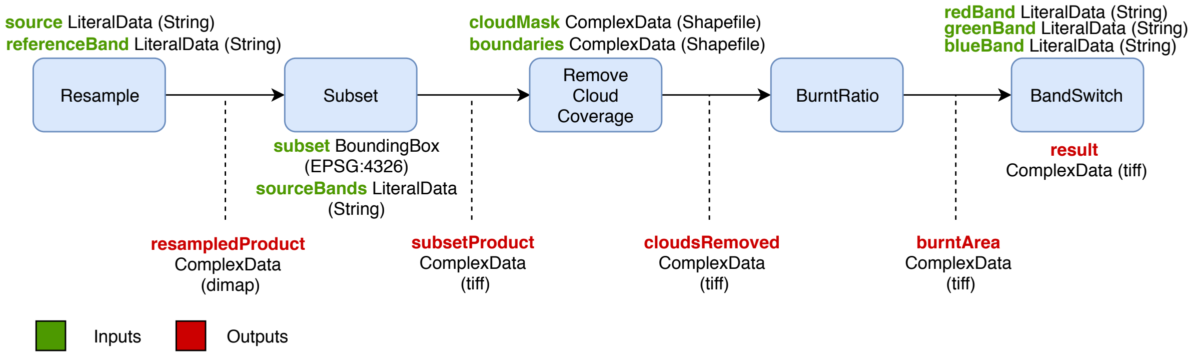

source LiteralData (String)
referenceBand LiteralData (String)
Resample
Subset
subset BoundingBox (EPSG:4326)
sourceBands LiteralData (String)
cloudMask ComplexData (Shapefile)
boundaries ComplexData (Shapefile)
Remove Cloud Coverage
BurntRatio
redBand LiteralData (String)
greenBand LiteralData (String)
blueBand LiteralData (String)
BandSwitch
result ComplexData (tiff)
resampledProduct ComplexData (dimap)
subsetProduct ComplexData (tiff)
cloudsRemoved ComplexData (tiff)
burntArea ComplexData (tiff)
Inputs
Outputs

Figure 4.1: Diagram representing the steps, 4.1.2, for wildfire identification and mapping, with the respective inputs and outputs.

These steps can be translated into web processing services, which can be found in appendix B. 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.

4.2 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 3. 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 3.2.3, since the combination, for example, in the ComplexData would be exponential. However, we will try to cover every data validation problem introduced in 2.2.3.

The workflow presented in 4.1.3, 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.

---

[3] Delineate the land boundaries of, per example, a country.

4.2.1 Expected Result

The final result should be the same as the one presented in figure 4.2, which was removed from EFFIS, where the burnt areas are highlighted in red.

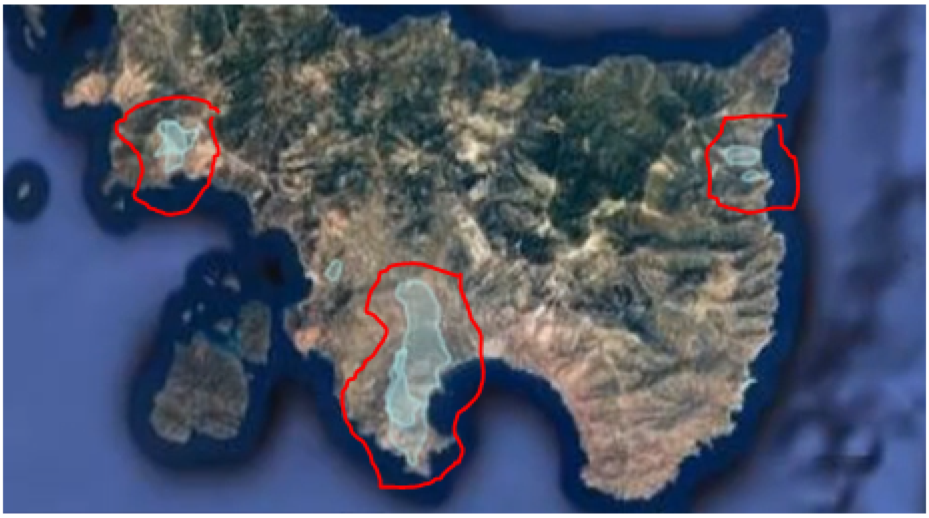


Figure 4.2: Wildfires that occurred in the municipality of Karistos during mid July and mid August of 2016.

Therefore, we consider that our solution is correct if it is able to: compose the workflow of WPSs 4.1.3, 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.

4.2.2 Workflow

The first step to test our solution is to compose the workflow schematic presented in 4.1.3, as it can be seen in figure 4.3.

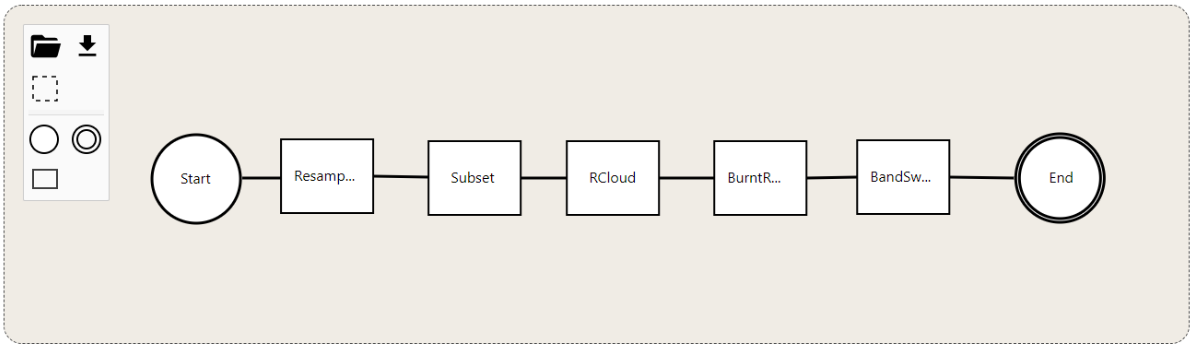


Figure 4.3: WPS-V-WEB workflow composition of the workflow schematic presented above, figure 4.1.

Since the workflow is composed, the next step is to fill in the inputs and outputs, but, instead of showing them one by one for each of the links, we will display the inputs directly through the internal workflow representation, listing 4.1

Inputs & Outputs

```
{
   ”identifier”:”d76183de−e82c−4d73−9949−2139ee231abb”,
   ”owner”:”0f6617f0−cb63−11e8−943c−8db390feb618”,
   ”chain”:[
      {
         ”index”:0,
         ”task”:{
            ”processURL”:”Resample.xml”,
            ”schema”:”http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd”,
            ”inputs”:{
               ”source”:[
                  {
                     ”sourceType”:”EXTERNAL”,
                     ”valueDto”:{
                        ”value”:”products/inputs/S2A_MSIL1C_20160829T090552_N0204_R050_T35SKC_20160829T090847.SAFE/MTD_MSIL1C.xml”,
                        ”dtoType”:”LiteralDataDto”
                     }
                  }
               ],
               ”referenceBand”:[
                  {
                     ”sourceType”:”EXTERNAL”,
                     ”valueDto”:{
                        ”value”:”B2”,
                        ”dtoType”:”LiteralDataDto”
                     }
                  }
               ]
            },
            ”outputs”:[
               ”resampledProduct”
            ]
         }
      },
      {
         ”index”:1,
         ”task”:{
            ”processURL”:”Subset.xml”,
            ”schema”:”http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd”,
            ”inputs”:{
               ”subset”:[
                  {
                     ”sourceType”:”EXTERNAL”,
                     ”valueDto”:{
                        ”crs”:”EPSG:4326”,
                        ”value”:”37.918 24.678 38.196 24.125”,
                        ”dtoType”:”BoundingBoxDataDto”
                     }
                  }
               ],
               ”product”:[
                  {
                     ”sourceType”:”INTERNAL”,
                     ”valueDto”:{
                        ”index”:0,
                        ”outputIdentifier”:”resampledProduct”,
                ”dtoType”:”InternalDataDto”
              }
            }
          ],
          ”sourceBands”:[
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B2”,
                ”dtoType”:”LiteralDataDto”
              }
            },
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B3”,
                ”dtoType”:”LiteralDataDto”
              }
            },
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B4”,
                ”dtoType”:”LiteralDataDto”
              }
            },
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B8”,
                ”dtoType”:”LiteralDataDto”
              }
            },
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B11”,
                ”dtoType”:”LiteralDataDto”
              }
            },
            {
              ”sourceType”:”EXTERNAL”,
              ”valueDto”:{
                ”value”:”B12”,
                ”dtoType”:”LiteralDataDto”
              }
            }
          ]
        },
        ”outputs”:[
          ”subsetProduct”
        ]
      }
    },
   (.......................................................................)
  ]
}
```

Listing 4.1: Internal representation of the workflow presented in 4.2.2 for the first two WPSs.

If we look at the listing above, we notice that there are two types of inputs according to the source type, internal and external, as it was said in 3. 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. With that in mind, now, we will discuss some possible validation problems that may occur in this chain, and how our solution solves them.

4.2.3 Validation Cases

In this subsection, we will demonstrate, for each of data validation problems in 2.2.3, 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[4], and insufficient data. Additionally, we can, also, detect occurrences validation. However, we do not have any case of redundant data, which is normal given that these WPSs only generate one output. But, even if we had, during 3 we have clarified that we only validate what is used in the workflow.

Mismatching

As it was said before 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.

Lets 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 91 38.196 24.125] results in an error, as it can be seen in figure 4.4. In the same way, by using Apache SIS we are able to support an extensive list[5] of coordinate reference systems, and standards such as WGS 84, NAVD 88, NAD 83, GRS 80, UTM, etc.

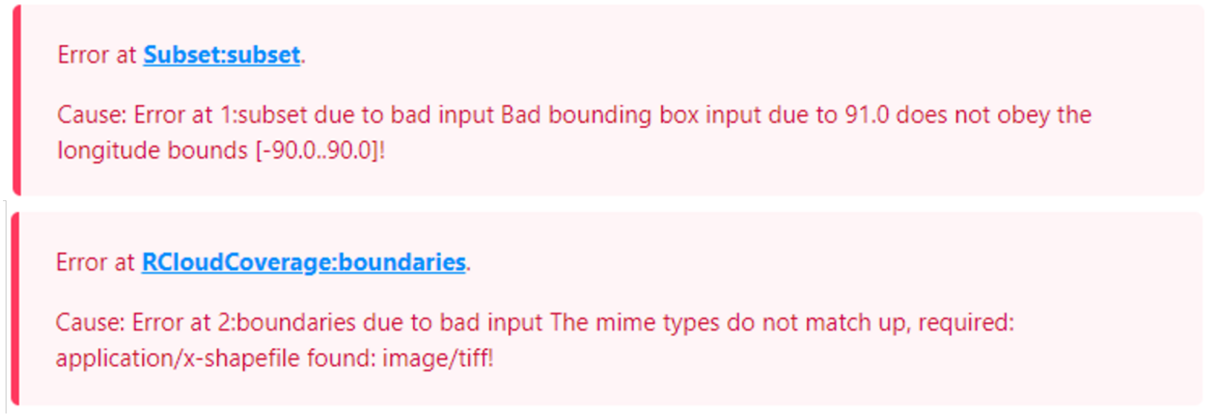
Lastly, we have **ComplexData**, which has a considerable representation in the workflow, for instance, we will look at the *boundaries* input in the RemoveCloudCoverage. It expects a vector shapefile representing a polygon of the administrative boundaries of Greece. Note that this process, for the area shown in figure 4.2, 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 3, 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

---

[4] Missing data usually can happen in almost any workflow

[5] `https://sis.apache.org/tables/CoordinateReferenceSystems.html`

file, for example a image/tiff, our solution picks it up, thanks to Apache Tika, as it can be seen in figure 4.4.



Error at **Subset:subset**.

Cause: Error at 1:subset due to bad input Bad bounding box input due to 91.0 does not obey the longitude bounds [-90.0..90.0]!

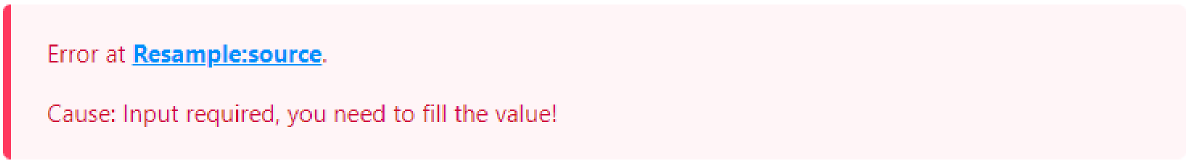
Error at **RCloudCoverage:boundaries**.

Cause: Error at 2:boundaries due to bad input The mime types do not match up, required: application/x-shapefile found: image/tiff!

Figure 4.4: Error messages thrown in the console in the WPS-V-WEB resulting from mismatching data for the BoundingBoxData (*subset*) and ComplexData (*boundaries*).

Missing Data

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 4.5, to alert the user.



Error at **Resample:source**.

Cause: Input required, you need to fill the value!

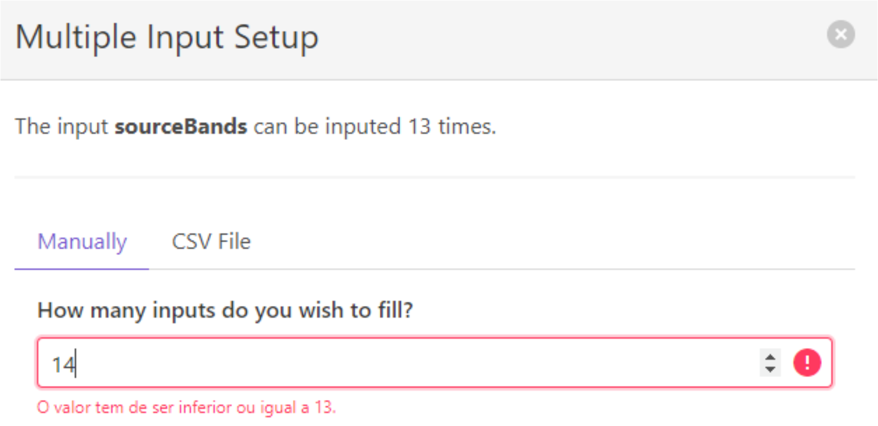
Figure 4.5: Error message thrown in the console in the WPS-V-WEB resulting from a missing input in the first WPS process (*source*).

Insufficient Data

Like all the other validation problems, insufficient data was already introduced before and 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 3.11. Actually, the simplest implementation is to abort the execution of the workflow, as we did for Camunda, through a rest call.

Occurrences Validation

The maximum number of occurrences for the *sourceBands* at the Subset WPS is 13[6], 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 4.6. Similarly, the WPS-V module also confirms this.


Multiple Input Setup

The input **sourceBands** can be inputed 13 times.

How many inputs do you wish to fill?

14

O valor tem de ser inferior ou igual a 13.

Figure 4.6: Number of bands supported in a sentinel-2 image restriction to the user in the multiple input setup.

4.2.4 Translation: Camunda (BPMN)

As you might remember from 3, 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 4.7, in order to be used on Camunda.

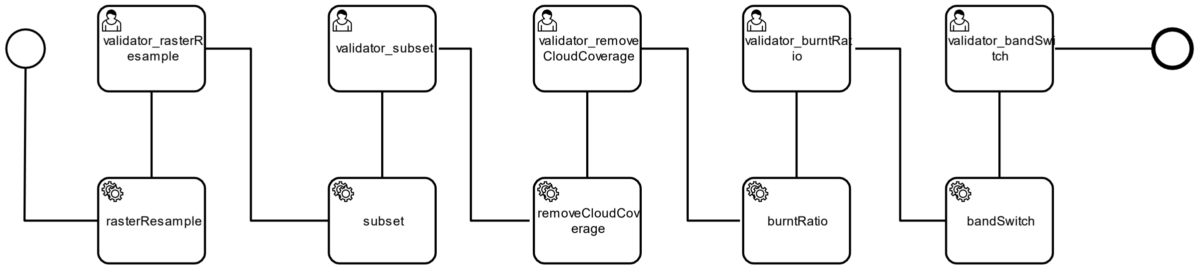


Figure 4.7: The workflow representation of 4.3 in Camunda, after the translation process.

It should be noted that we could have chosen any of WMSs presented in 2.3. Nevertheless, Camunda provides: rest endpoints for every important control operation, such as deploy and

---

[6] `https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/resolutions/spatial`

abort, an easy installation, and support for BPMN. Moreover, Camunda is gaining popularity in EO, [3] and [30].

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 an user task.

4.2.5 Results

The results from executing the workflow, figure 4.3, in Camunda, are represented in figure 4.8.

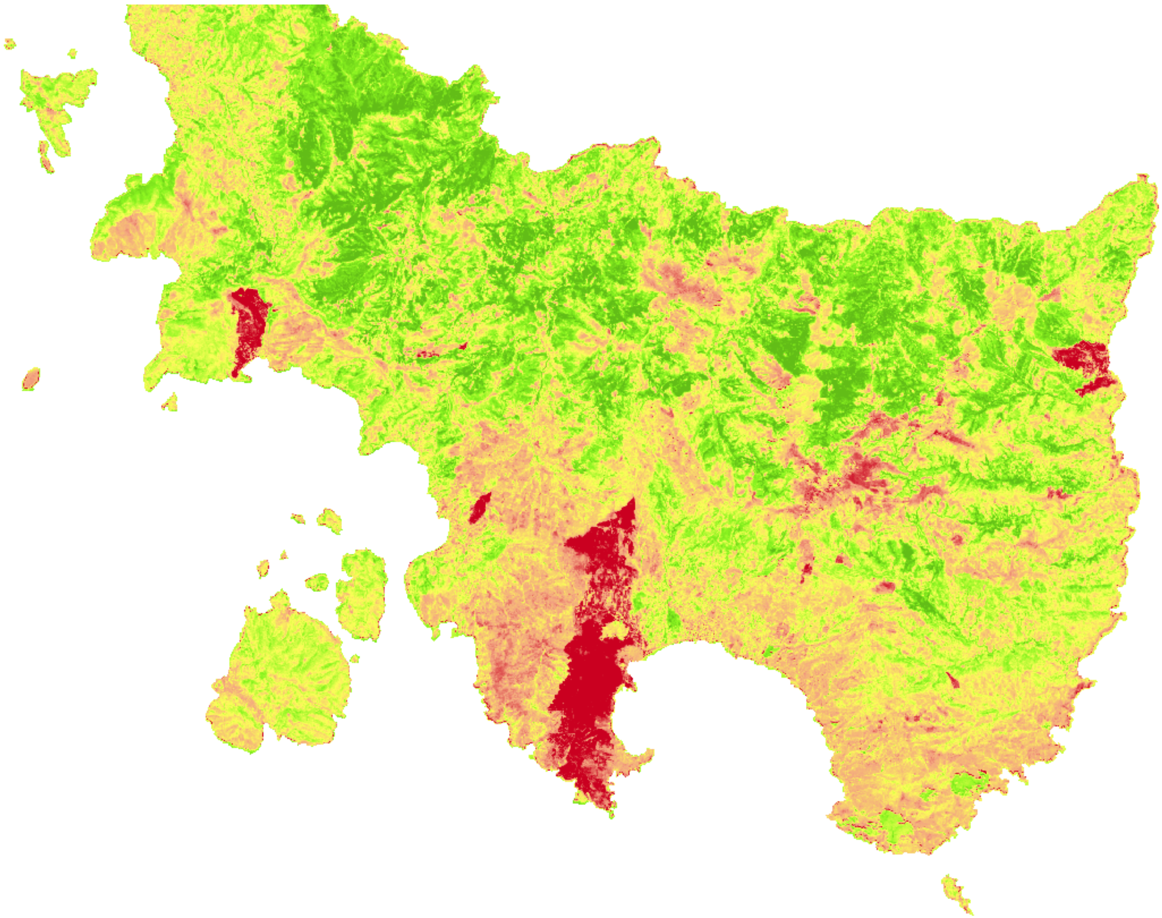


Figure 4.8: Greece burnt areas in mid July to mid August of 2016, identified and mapped by the workflow of WPSs presented in 4.3.

The strong red areas are the burnt areas, which correspond to the ones expected in 4.2.1. Thus, two things were proven:

1. The workflow of WPSs presented in 4.3 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.

4.2.6 Final Considerations

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

We have found during our investigation in platforms, such as tep[7] and co-resyf[8], 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, [34], 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.

With this in mind, and with the results presented, we established and proven a solution, where, there is, clearly, an opportunity to disaggregate these services into smaller ones.

---

[7] `https://tep.eo.esa.int/`

[8] `http://co-resyf.eu/`

Chapter 5

Conclusions

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.

5.1 Achievements

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[1] - **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.

5.2 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,

---

[1] It should be capable of using web services and a specific implementation should be done to provide translation to it.

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 chapter 3, 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, [30], which will be used to deploy and undeploy WPSs in DCIs. 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 this problems.

Bibliography

[1] P. Kansakar and F. Hossain. A review of applications of satellite earth observation data for global societal benefit and stewardship of planet earth. *Space Policy*, 36:46–54, 2016. ISSN 1879338X. doi: 10.1016/j.spacepol.2016.05.005. URL `http://dx.doi.org/10.1016/j.spacepol.2016.05.005`.

[2] R. Buyya, C. S. Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering IT services as computing utilities. *Proceedings - 10th IEEE International Conference on High Performance Computing and Communications, HPCC 2008*, pages 5–13, 2008. ISSN 978-0-7695-3352-0. doi: 10.1109/HPCC.2008.172.

[3] OGC. OGC Testbed-13: Workflows ER. Technical report, . URL `http://docs.opengeospatial.org/per/17-029r1.html{#}{_}summary`.

[4] S. Falke, E. Dorner, B. Dunn, and D. Sullivan. Processing Services in Earth Observation Sensor Web Information Architectures. *Forecast.*

[5] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger. Data Flow and Validation in Workflow Modelling. *Adc'04*, 27:207–214, 2004. URL `http://portal.acm.org/citation.cfm?id=1012294.1012317`.

[6] C. S. Liew, M. P. Atkinson, M. Galea, T. F. Ang, P. Martin, and J. I. Van Hemert. Scientific Workflows: Moving Across Paradigms. *ACM Computing Surveys*, 49(4):66, 2017. ISSN 0360-0300. doi: 10.1145/0000000.0000000.

[7] A. Bosin, N. Dessì, and B. Pes. Extending the SOA paradigm to e-Science environments. *Future Generation Computer Systems*, 27(1):20–31, 2011. ISSN 0167739X. doi: 10.1016/j.future.2010.07.003. URL `http://dx.doi.org/10.1016/j.future.2010.07.003`.

[8] M. Fowler and J. Lewis. Microservices a definition of this new architectural term. URL `https://martinfowler.com/articles/microservices.html`.

[9] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006. ISSN 1532-0626. doi: 10.1002/cpe.994. URL `http://doi.wiley.com/10.1002/cpe.994`.

[10] N. Chen, L. Di, G. Yu, and J. Gong. Geo-processing workflow driven wildfire hot pixel detection under sensor web environment. *Computers and Geosciences*, 36(3):362–372, 2010. ISSN 00983004. doi: 10.1016/j.cageo.2009.06.013. URL `http://dx.doi.org/10.1016/j.cageo.2009.06.013`.

[11] Liping Di. GeoBrain-A Web Services based Geospatial Knowledge Building System. *Proceeding of NASA Earth Science Technology Conference, 2004 June 22 - 24, Palo Alto CA USA*, page 8, 2004.

[12] G. Hobona, D. Fairbairn, H. Hiden, and P. James. Orchestration of grid-enabled geospatial Web services in geoscientific workflows. *IEEE Transactions on Automation Science and Engineering*, 7(2):407–411, 2010. ISSN 15455955. doi: 10.1109/TASE.2008.2010626.

[13] B. Stollberg and A. Zipf. OGC Web Processing Service Interface for Web Service Orchestration Aggregating Geo-processing Services in a Bomb Threat Scenario. In J. M. Ware and G. E. Taylor, editors, *Web and Wireless Geographical Information Systems*, pages 239–251, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76925-5.

[14] F. Bensmann, D. Alcacer-Labrador, D. Ziegenhagen, and R. Roosmann. The RichWPS Environment for Orchestration. *ISPRS International Journal of Geo-Information*, 3(4): 1334–1351, 2014. ISSN 2220-9964. doi: 10.3390/ijgi3041334. URL `http://www.mdpi.com/2220-9964/3/4/1334/`.

[15] S. Schade, N. Ostländer, C. Granell Canut, M. Schulz, D. McInerney, G. Dubois, L. Vaccari, M. Chinosi, L. Sánchez, Díaz, L. Bastin, and R. Jones. Which Service Interfaces fit the Model Web ? *GEOProcessing 2012: The Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services*, (c):1–6, 2012. URL `http://repositori.uji.es/xmlui/handle/10234/159973`.

[16] D. Alcacer-labrador, B. Sc, F. Bensmann, and M. Sc. 09-Alcacer-Labrador_0.pdf. URL `http://europe.foss4g.org/2014/sites/default/files/09-Alcacer-Labrador{_}0.pdf`.

[17] A. Pratt, C. Peters, S. Guru, B. Lee, and A. Terhorst. Exposing the Kepler scientific workflow system as an OGC web processing service. *Mod-*

elling for Environment's Sake: Proceedings of the 5th Biennial Conference of the International Environmental Modelling and Software Society, iEMSs 2010, 2: 1554–1561, 2010. URL `http://www.scopus.com/inward/record.url?eid=2-s2.0-84863363130{&}partnerID=40{&}md5=4db1e6494a4c212199cd449b69297622`.

[18] G. Jiménez, R. Béjar, M. Latre, and P. Muro-Medrano. A Method to Derivate SOAP interfaces and WSDL Metadata from OGC Web Processing Service Mandatory Interfaces. URL `https://dl.acm.org/citation.cfm?id=1478056`.

[19] S. Meek. OGC 16-091 BPMN 2.0 for Orchestrating OGC Services,. Technical report. URL `https://portal.opengeospatial.org/files/?artifact{_}id=68879{&}version=1`.

[20] J. Rosser, A. Pourabdollah, R. Brackin, M. Jackson, and D. Leibovici. Full Meta Objects for flexible geoprocessing workflows: profiling WPS or BPMN? *The 19th AGILE International Conference on Geographic Information Science*, (June), 2016.

[21] P. Dana, P. Silviu, N. Marian, F. Marc, Z. Daniela, and C. Radu. Earth Observation Data Processing in Distributed Systems 1 Introduction 2 EO requests on distributed environments. *Informatica*, 34(4):463–476, 2010.

[22] E. Deelman, D. Gannon, M. Shields, and I. Taylor. Workflows and e-Science: An overview of workflow system features and capabilities. *Future Generation Computer Systems*, 25(5): 528–540, 2009. ISSN 0167739X. doi: 10.1016/j.future.2008.06.012. URL `http://dx.doi.org/10.1016/j.future.2008.06.012`.

[23] S. Shahrivari. Beyond Batch Processing: Towards Real-Time and Streaming Big Data. *Computers*, 3(4):117–129, 2014. ISSN 2073-431X. doi: 10.3390/computers3040117. URL `http://www.mdpi.com/2073-431X/3/4/117/`.

[24] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40 (12):24–32, 2007. ISSN 00189162. doi: 10.1109/MC.2007.421.

[25] Y. Gil, V. Ratnakar, J. Kim, J. Moody, E. Deelman, P. A. González-calero, U. C. D. Madrid, P. Groth, and V. U. Amsterdam. Wings : Intelligent Experiments. *IEEE Intelligent Systems*.

[26] Y. Gil, V. Ratnakar, and C. Fritz. Assisting {S}cientifits with {C}omplex {D}ata {A}nalysis {T}asks through {S}emantic {W}orkflow. *Proceedings of the 2010 {F}all {S}ymposium on {P}roactive {A}ssistant {A}gents*, (November), 2010.

[27] Y. Gil, V. Ratnakar, E. Deelman, M. Spraragen, and J. Kim. Wings for pegasus: A semantic approach to creating very large scientific workflows. *CEUR Workshop Proceedings*, 216, 2006. ISSN 16130073.

[28] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic acids research*, 41(Web Server issue):557–561, 2013. ISSN 13624962. doi: 10.1093/nar/gkt328.

[29] J. de Jesus, P. Walker, M. Grant, and S. Groom. WPS orchestration using the Taverna workbench: The eScience approach. *Computers and Geosciences*, 47:75–86, 2012. ISSN 00983004. doi: 10.1016/j.cageo.2011.11.011. URL `http://dx.doi.org/10.1016/j.cageo.2011.11.011`.

[30] OGC. Testbed 14. Technical report, . URL `https://portal.opengeospatial.org/files/77326`.

[31] Y. Shao, L. Di, Y. Bai, B. Guo, and J. Gong. Geoprocessing on the Amazon cloud computing platform AWS. *2012 1st International Conference on Agro-Geoinformatics, Agro-Geoinformatics 2012*, pages 286–291, 2012. doi: 10.1109/Agro-Geoinformatics.2012.6311655.

[32] Baranski, B.Schaeffer, and B. Redweik. Geoprocessing in the Clouds. *OSGeo Journal - 8, 1, 5*, 2010.

[33] B. Mclaughlin. Validation with Java and XML Schema, 2000. URL `https://www.javaworld.com/article/2076170/java-se/validation-with-java-and-xml-schema`.

[34] F. J. Lopez-Pellicer, W. Rentería-Agualimpia, R. Béjar, P. R. Muro-Medrano, and F. J. Zarazaga-Soria. Availability of the OGC geoprocessing standard: March 2011 reality check. *Computers and Geosciences*, 47(March 2011):13–19, 2012. ISSN 00983004. doi: 10.1016/j.cageo.2011.10.023.

Appendix A

WPS Operations Detailed

A.1 GetCapabilities

The GetCapabilities operation requests details of the service offering, including service metadata and metadata describing the available processes. The response is an XML document called the capabilities document. An example of a GetCapabilities request is given by:

`http://localhost:5000/wps?service=wps&request=getcapabilities`

The result of the following request is:

```
<wps:Capabilities xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ows="http://www.opengis.net/ows/1.1"xmlns:gml="http://www.opengis.net/gml" service="WPS" version="1.0.0" xml:lang="en-US" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd"updateSequence="1">
  <ows:ServiceIdentification>...</ows:ServiceIdentification>
  <ows:ServiceProvider>...</ows:ServiceProvider>
  <ows:OperationsMetadata>
   <ows:Operation name="GetCapabilities">...</ows:Operation>
   <ows:Operation name="DescribeProcess">...</ows:Operation>
   <ows:Operation name="Execute">...</ows:Operation>
  </ows:OperationsMetadata>
  <wps:ProcessOfferings>
   <wps:Process wps:processVersion="1.0.0">
     <ows:Identifier>calibration</ows:Identifier>
     <ows:Title>Calibration Process</ows:Title>
     <ows:Abstract>Calibrates a level 1 product</ows:Abstract>
   </wps:Process>
   <wps:Process wps:processVersion="1.0.0">
```

```
      <ows:Identifier>subset</ows:Identifier>
      <ows:Title>Subset</ows:Title>
      <ows:Abstract>Process returns the subset a GML file</ows:Abstract>
    </wps:Process>
    <wps:Process wps:processVersion="1.0.0">
      <ows:Identifier>oilspilldetection</ows:Identifier>
      <ows:Title>Oil Spill Detection</ows:Title>
      <ows:Abstract>Detect oil spill detection over a image/tiff</ows:Abstract>
    </wps:Process>
  </wps:ProcessOfferings>
  <wps:Languages>...</wps:Languages>
</wps:Capabilities>
```

Listing A.1: GetCapabilities request example.

Analyzing the GetCapabilities response, we can identify the operations supported, OperationMetadata, and the processes offered by the WPS, processOfferings, which are the same as the 2.1. Note that this WPS offers three processes, but they act as independent web services, this is, they just happen to run under the same machine and web server. The rest of the tags, ServiceIdentification and ServiceProvider, provides important metadata about the service (WPS version running, an abstract about the purpose of the service, etc) and the provider (entity providing the service and information about the developer, like name, phone, address, etc).

A.2 DescribeProcess

The DescribeProcess operation allows WPS clients to request a full description of one process executed by the service. This description, a XML document, includes the input and output parameters and formats and can be used. Taking the calibration process as an example, a DescribeProcess request to it is given by:

`http://localhost:5000/wps?service=wps&request=describeprocess&identifier=calibration&version=1.0.0`

The XML response:

```xml
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml"xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
 <ProcessDescription wps:processVersion="1.0.0" storeSupported="true" statusSupported="true">
   <ows:Identifier>calibration</ows:Identifier>
   <ows:Title>Calibration Process</ows:Title>
   <ows:Abstract>Calibrates a level 1 product</ows:Abstract>
   <DataInputs>
     <Input minOccurs="1" maxOccurs="1">
       <ows:Identifier>product_url</ows:Identifier>
       <ows:Title>ProductUrl</ows:Title>
       <LiteralData>
         <ows:DataType ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
         <ows:AnyValue/>
       </LiteralData>
     </Input>
   </DataInputs>
   <ProcessOutputs>
     <Output>
       <ows:Identifier>calibrated_image</ows:Identifier>
       <ows:Title>CalibratedImage</ows:Title>
       <LiteralOutput>
         <ows:DataType ows:reference="urn:ogc:def:dataType:OGC:1.1:string">string</ows:DataType>
       </LiteralOutput>
     </Output>
   </ProcessOutputs>
 </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing A.2: DescribeProcess of Calibration process.

From the XML response we can see that the process Calibration, calibrates a level 1 product and has service version 1.0.0. Furthermore, in DataInputs we can see that this process receives as input a LiteralData of type string (product_url), and, in ProcessOutputs, outputs, also, a LiteralData of type string (calibrated_image). These strings, in this concrete case, represent urls to products, which may be hosted locally or remotely. Note that, we could have defined the inputs and outputs as ComplexData types, per example, in Geographic Markup Language (GML). But, for sake of diversity and further conclusions, we opted for a string representation. Thus, inputs and outputs may be different between processes and both offer the same functionality.

A.3 Execute

The Execute operation allows WPS clients to run a specified process implemented by a server, using the input parameter values provided and returning the output values produced. Inputs can be included directly in the Execute request, or reference web accessible resources[1]. Taking the calibration process as an example, a Execute request to it is given by:

```
http://localhost:5000/wps?service=wps&request=execute&identifier=calibration&
version=1.0.0&datainputs=product_url=http%3A%2F%2Flocaldomain%2Fproduct
```

In this request, we can see the identifier (product_url) retrieved by the DescribeProcess operation taking the value of a string, which is a url. Inputs are separated through semicolons. The corresponding response, in XML, of the previous request is:

```
<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.opengis.net/gml"xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US" serviceInstance="http://localhost/wps?service=WPS&request=GetCapabilities"statusLocation="file:///tmp/e3d35c10-d757-11e7-a39f-0800276307ac.xml">
 <wps:Process wps:processVersion="1.0.0">
   <ows:Identifier>calibration</ows:Identifier>
   <ows:Title>Calibration Process</ows:Title>
   <ows:Abstract>Calibrates a level 1 product</ows:Abstract>
 </wps:Process>
 <wps:Status creationTime="2017-12-02T11:56:54Z">
   <wps:ProcessSucceeded>Process Calibration Process finished</wps:ProcessSucceeded>
 </wps:Status>
 <wps:ProcessOutputs>
   <wps:Output>
     <ows:Identifier>calibrated_image</ows:Identifier>
     <ows:Title>CalibratedImage</ows:Title>
```

---

[1] `http://geoprocessing.info/wpsdoc/1x0Execute`

```
      <wps:Data>
        <wps:LiteralData dataType="urn:ogc:def:dataType:OGC:1.1:string">http://localdomain/
            product.tiff</wps:LiteralData>
      </wps:Data>
    </wps:Output>
  </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

Listing A.3: Execute of the Calibration process.

The Execute operation is similar to the DescribeProcess, with two particular differences. The DataInputs disappear and the ProcessOutputs in the LiteralData, callibrated_image, has the output url from the transformation execution. This url can now be used as input to the next process in the chain, in our example Subset, given that it accepts as input a LiteralData (string) that will be interpreted as an url.

Appendix B

Wildfire WPSs

B.1 Resample Raster

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
    <ProcessDescription wps:processVersion="None" storeSupported="true" statusSupported="true">
        <ows:Identifier>rasterResample</ows:Identifier>
        <ows:Title>Resample a product from a reference band</ows:Title>
        <ows:Abstract>Resample a product from a reference band</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>source</ows:Identifier>
                <ows:Title>Source Product URL</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>referenceBand</ows:Identifier>
                <ows:Title>Band to be used as the resampling reference</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>resampledProduct</ows:Identifier>
                <ows:Title>Resampled Product</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>image/x-dimap</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/x-dimap</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing B.1: Resample Raster Describe Process.

B.2 Subset

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
    <ProcessDescription wps:processVersion="None" storeSupported="true" statusSupported="true">
        <ows:Identifier>subset</ows:Identifier>
        <ows:Title>Subset a product according to an area of interest</ows:Title>
        <ows:Abstract>Subset a product according to an area of interest</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>subset</ows:Identifier>
        <ows:Title>Bounding</ows:Title>
        <BoundingBoxData>
          <Default>
            <CRS>EPSG:4326</CRS>
          </Default>
          <Supported>
            <CRS>EPSG:4326</CRS>
          </Supported>
        </BoundingBoxData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>product</ows:Identifier>
                <ows:Title>Product to subset</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexData maximumMegabytes="1">
                    <Default>
                        <Format>
                            <MimeType>image/x-dimap</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/x-dimap</MimeType>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
            <Input minOccurs="1" maxOccurs="13">
                <ows:Identifier>sourceBands</ows:Identifier>
                <ows:Title>List of Bands to include in the subset</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</
                    ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
```

```
            <Output>
                <ows:Identifier>subsetProduct</ows:Identifier>
                <ows:Title>Subset Product</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing B.2: Subset Describe Processs.

B.3 Remove Cloud Coverage

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
    <ProcessDescription wps:processVersion="None" storeSupported="true" statusSupported="true">
        <ows:Identifier>removeCloudCoverage</ows:Identifier>
        <ows:Title>Remove Cloud Coverage from an area of interest</ows:Title>
        <ows:Abstract>Remove Cloud Coverage from an area of interest</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>product</ows:Identifier>
                <ows:Title>Product to Remove Cloud Coverage</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexData maximumMegabytes="1">
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>cloudMask</ows:Identifier>
                <ows:Title>Cloud mask shapefile</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexData maximumMegabytes="1">
                    <Default>
                        <Format>
                            <MimeType>application/x-shapefile</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>application/x-shapefile</MimeType>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>boundaries</ows:Identifier>
                <ows:Title>Administrative Boundaries Shapefile</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexData maximumMegabytes="1">
                    <Default>
                        <Format>
                            <MimeType>application/x-shapefile</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>application/x-shapefile</MimeType>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>noCloudsProduct</ows:Identifier>
                <ows:Title>Product with no clouds</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing B.3: Remove Cloud Coverage Describe Process.

B.4 Burnt Ratio

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
    <ProcessDescription wps:processVersion="None" storeSupported="true" statusSupported="true">
        <ows:Identifier>burntRatio</ows:Identifier>
        <ows:Title>Calculate the burn ratio on an area of interest</ows:Title>
        <ows:Abstract>Calculate the burn ratio on an area of interest</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
```

```xml
                <ows:Identifier>product</ows:Identifier>
                <ows:Title>Product to Calculate the Burnt Ratio</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexData maximumMegabytes="1">
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>burntArea</ows:Identifier>
                <ows:Title>Product with burnt area</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing B.4: Burnt Ratio Describe Process.

B.5 Band Switch

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 ../wpsDescribeProcess_response.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
    <ProcessDescription wps:processVersion="None" storeSupported="true" statusSupported="true">
        <ows:Identifier>bandSwitch</ows:Identifier>
        <ows:Title>Switch the bands of a product</ows:Title>
        <ows:Abstract>Switch the bands of a product to emphasize some characteristic</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>redBand</ows:Identifier>
                <ows:Title>Red Band Number</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>greenBand</ows:Identifier>
                <ows:Title>Green Band Number</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>blueBand</ows:Identifier>
                <ows:Title>Blue Band Number</ows:Title>
                <ows:Abstract></ows:Abstract>
                <LiteralData>
                <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                <ows:AnyValue />
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>output</ows:Identifier>
                <ows:Title>Product resulting from switching the bands</ows:Title>
                <ows:Abstract></ows:Abstract>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>image/tiff</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing B.5: Band Switch Describe Process.