DESIGN AND EVALUATE THE OGC WEB SERVICES ARCHITECTURE OF A GEOHAZARD SUSCEPTIBILITY ANALYSIS TOOL

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Ioanna Micha

October 2019

The work in this thesis was carried out in the:

Supervisors:	Prof.dr.ir. P.J.M. van Oosterom
	Drs. ME de Vries
	Drs. Ing. Gerrit Hendriksen (Deltares), Mike Woning (Deltares)
Co-reader:	Dr.ir. P. (Pirouz) Nourian

ABSTRACT

Open standards like Open Geospatial Consortium (OGC) standards can be used to improve interoperability and re-usability among geospatial tools, datasets and services across the web. Today more than six thousands commercial and non commercial implementations of the OGC Web Services (OWS) are available, and more and more Geographical Information Systems (GIS) web applications, platforms and datasets are being published from organizations in the global geospatial community. This thesis explores if the GIS standards, their implementations, and the organizations themselves are mature enough to support quality decision making in sectors like the risk management, by developing GIS web applications with *up-to-date* spatial datasets and complex *geo-processes*.

Motivation for this research has been the Risk Indicators for Infrastructure in Data scarce Environments (RI2DE) GIS web tool of Deltares institution, a tool that performs GIS analysis based on datasets with global coverage, in order to detect areas around road infrastructures that are susceptible against climate related geohazards, and its need to provide quality risk maps and more friendly to the user Graphical User Interface (GUI) through status reporting messages and control of the processes. The tool is developed with open source technology components, OWS and standard data formats (such us Geographic JavaScript Object Notation (JSON) (GeoJSON), JSON and eXtensible Markup Language (XML).

Having as case study the RI2DE tool, within this research the OGC Web Processing Service (WPS) 2.0.2 standard that is responsible for publishing geoprocesses as web services and its free and open source *PyWPS* 4.0.0 implementation has been evaluated according to the new operations for *job monitoring* and *job control* that the standard offers, in order to assess if it is feasible to have status report messages and control of the processes. Moreover it has be assessed how easy is to *find* and *access up-to-date* datasets, services and tools from distributed Spatial Data Infrastructures (SDI)s that implement the Catalogue Service for the Web (CSW) standard, and how flexible are the WPS to accept different inputs.

From an early stage of this research it has been clear that the status reporting and the control of the process would have been difficult to be implemented, since the *PyWPS* implementation of the WPS 2.0.2 is still undergoing until the delivery of this thesis. While the standard has been published since 2015, the only available full implementation of it is from the *ZOO-Project*, which arise question regarding the complexity of the standard specification and the currently state-of-the-art technology.

The greatest achievement of this thesis has been the integration of the distributed searching for services at different CSW catalogues, at the RI2DE tool web browser. While the implementation was achieved, many question were arise concerning the sharing of the produced geospatial information from organizations and countries.

ACKNOWLEDGEMENTS

I would like first to thank my supervisor from Deltares, Drs Gerrit Hendriksen for always being there for me and also for providing the idea for this research.

I would also like to thank my thesis supervisors, Prof.dr.ir. Peter van Oosterom and Drs. Marianne de Vries for their guidance and their ideas on the research.

I would also like to acknowledge Dr.ir. Pirouz Nourian as the co-reader of this thesis, for his very valuable comments.

I would also like to thank my other supervisor from Deltares, Mike Woning for providing the RI2DE tool for this research.

Also, sincere thanks to my family and especially to my sister for their love and support these two years that I am abroad.

Last, but not least, I need to thank my friends, here in Netherlands who have always been there for me !!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DEM Digital Elevation Model . . . 2
GIS Geographical Information Systems . . . iii
GUI Graphical User Interface . . . iii
W3C World Wide Web Consortium . . . 1
WCS Web Coverage Service . . . 3
WPS Web Processing Service . . . iii
WMS Web Map Service . . . 3
CSW Catalogue Service for the Web . . . iii
OWS OGC Web Services . . . iii
OGC Open Geospatial Consortium . . . iii
XML eXtensible Markup Language . . . iii
JSON JavaScript Object Notation . . . iii
GeoJSON Geographic JSON . . . iii
GML Geography Markup Language . . . 1
ISO International Organization for Standardization . . . xiii
CRS Coordinate Reference System . . . 32
EPSG European Petroleum Survey Group . . . 62
WGS84 World Geodetic System 1984 . . . 52
HTTP Hypertext Transfer Protocol . . . 1
ISO/TC International Organization for Standardization (ISO) Technical Committee . 1
URL Uniform Resource Locator . . . 2
RI2DE Risk Indicators for Infrastructure in Data scarce Environments . . . iii
DSR Design Science Research . . . 5
ROADAPT VA Roads for today, adapted for tomorrow Vulnerability Assessment . . . 2
API Application Programming Interface . . . 28
SOAP Simple Object Access Protocol . . . 9
KVP Key-Value Pair . . . 9
KML Keyhole Markup Language . . . 22
OSM Open Street Map . . . 34
SVN SubVersion Repository . . . 35
MERIT Multi-Error Removed Improved Terrain . . . 36
SRTM Shuttle Radar Topography Mission . . . 36
ISRIC International Soil Reference and Information Centre . . . 37
ESA European Space Agency . . . 37
JRC Joint Research Centre . . . 40
SDI Spatial Data Infrastructures . . . iii
EU European Union . . . 30
FAST Foreshore Assessment using Space Technology . . . 30
SVG Scalable Vector Graphics . . . 10
CQL Common Query Language . . . 18
BNF Backus Naur Form . . . 18

1 INTRODUCTION

Geospatial data is an increasingly important information asset for decision making process. GIS are being used in many sectors, such as transportation, environmental monitoring and risk management. More *reliable* and *up-to date* spatial data and tools are required for proper decision making. Worldwide, governments at different levels and organizations have put a lot of effort into *disseminating* geo-spatial data and applications of GIS via the web for wide *reuse* van den Brink [2018]. But as more geographic data and related GIS web applications become available, organizations have been formed to answer the problem of *interoperability* between data and processes. International organizations within the geo-spatial community set standards and specifications for spatial data formats, metadata and services in order to reduce misunderstanding, to optimize operations, to improve the quality and promote *re-usability* of GIS tools and products Gistandards [2018].

Within this context the OGC, a non for profit organization that started in 1994 as the OpenGIS project, sets as goal to make quality open communication standards for the global geospatially community. To ensure consistency across the internet and web ecosystem, the OGC has alliance partnerships with many other standards development organizations, like the World Wide Web Consortium (W_3C) (founded in 1994 by Tim Berners-Lee and offers a variety of standards like XML for encoding documents) and the ISO Technical Committee (ISO/TC) 211 (formed within ISO in order to provide standards that cover the area of digital geographic information, like the ISO 19115 and ISO 19139 for the context and the schema of the Geographic information metadata) OGC [2019b].

On that terms OGC offers standard interfaces and encodings for spatial data formats (like the Geography Markup Language (GML), an XML representation for geographical features) and geo-services (the OWS). OWS are self-contained, self-describing, modular geo-spatial functionalities for data access, data display and data processing, that can be published and invoked across the web, using familiar communication technologies and data encodings, such as Hypertext Transfer Protocol (HTTP), JSON, and XML.

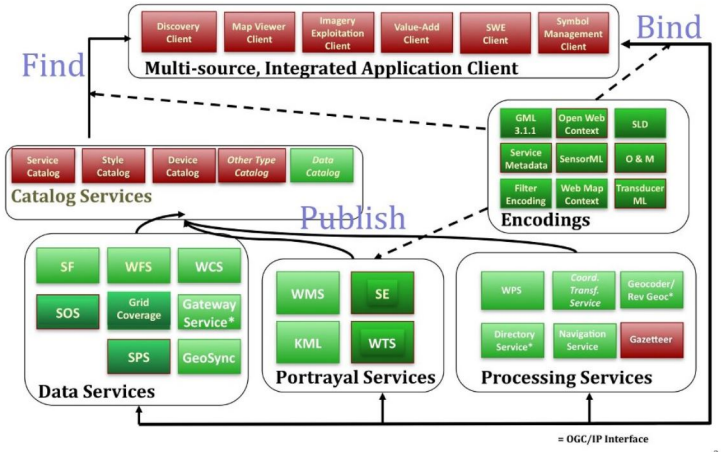

Figure 1.1: Web Services Framework of OGC Geoprocessing standards, Source: OSGeo

Within OGC vision it is possible to build complex GIS web applications, based on an OGC Web Services Oriented Architecture, where the components are free,

dynamic and web distributed services. In Figure 1.1 it is depicted a general architectural schema of the different OGC Web Services and standard encodings. The *publish-find-bind* paradigm describes the relations and functions that the OGC standards and services can offer. In the paradigm, OGC services providers publish their spatial datasets and processes as OWS services in public registries. These services are *accessible* by an *endpoint* (Uniform Resource Locator (URL) address), and geo-spatial organizations and consumers can find them and *reuse* them at any time.

More than six thousands approved and not approved OGC implementation standards are available in the geospatial community, either commercial or free and open source OGC [2019a], and more and more GIS web geoprocessing products are being developed from organizations and institutions around the world.

The idea of this thesis is to *evaluate* if the *standards*, their *implementations* and the *organizations/institutions* that produce spatial (datasets and tools), are mature enough to support the development of GIS web applications that demands *up-to-date spatial datasets* and *complex geo-processes*, for quality decision making. Motivated and having as case study the same time, the RI2DE GIS Web tool of Deltares institution, that is used for susceptibility analysis in areas around road infrastructures against climate triggered geohazards, and that is developed with free and open source technology components in an OGC Web Services environment, this thesis will explore:

- The OGC CSW standard that publish catalogues of geospatial records (metadata about gespatial data or services) and the ISO and Dublin core metadata schemas that are used for the registration of the different records, in order to assess how easy is to *find* and *access up-to-date* datasets and services, and how mature are the organizations on publishing and registering the datasets they produce.
- The OGC WPS 2.0.2 version of the standard that publish geospatial processing services as web services and the newly developed free and open source PyWPS 4.0.0 implementation of it, in order to assess what is its status regarding the *GetStatus*, *GetResult* and *Dismiss* operations in an asynchronous execution of the process.

1.1 MOTIVATION AND CASE STUDY

Modern societies are highly depended on their network infrastructures for their social and economic activities and disruptions in one of these infrastructures may have serious consequences. Depending on the local site environment and on the extreme weather conditions, different geohazards can potentially threaten the network infrastructure integrity and serviceability. Within the Deltares institution the RI2DE GIS Web tool that helps to determine the susceptibility of infrastructures (e.g. a road) for different climate triggered geo-hazards (e.g landslides) that affects it, has been developed on the GIS based Roads for today, adapted for tomorrow Vulnerability Assessment (ROADAPT VA) methodology. In this methodology various types of available information, such as Digital Elevation Model (DEM) and land use maps, become the susceptibility factors, and when they are combined they produce vulnerable locations against specific geohazards.

RI2DE tool base its GIS analysis on *global* coverage datasets with *low resolution* and, is developed and published as a project of the OpenEarth initiative of Deltares institution. OpenEarth is an initiative to deal with Data, Models and Tools in earth science engineering projects. To be an effective and sustainable paradigm solution in the development of GIS web products, OpenEarth adapts international standards for exchange of data and tools over the web, such as OGC and ISO and development of the OpenEarth tools, datasets and services is based mainly on *free and open source* technologies van Koningsveld and den Heijer [2014]. Being a project of this initiative RI2DE main components are OGC web services, and known GIS data formats are

used for the exchange of the information between the client and the server, such as GeoJSON and JSON. Also free and open source GIS packages and services' implementations are used. More specifically:

- All the geoprocesses that assess the susceptibility of the infrastructures are published as WPS
- The input datasets to the geoprocesses are transnational Web Coverage Service (WCS)
- The temporary results and the final vulnerability maps are published as Web Map Service (WMS) at the RI2DE *GeoServer* for visualization purposes
- The implementation of the WPS is based on the free and open source *PyWPS 4.0.0* implementation

The target audience of the RI2DE GIS web tool is either road infrastructure owners or governmental/local stakeholders that are deeply involved with the construction and conservation of national or local infrastructure networks. For them it is essential to conduct quality decision making regarding the road segments that are vulnerable to geohazards, in order to take appropriate preventative measures and perform good cost benefit analysis.

So far the input datasets to the GIS analysis of the tool are transnational services with low resolution. The need for quality decision making makes it necessary to use apart from the predefined services, higher quality and resolution services that derive from regional or national databases, and to have final vulnerability assessment on the road segments instead of simple raster maps. Increasing the resolution of the datasets, changing the sources, and having more complex processes might arise issues concerning the processing time or cause incompatibility errors that could collapse the web tool. Based on these needs there is a vision for a new version of the tool where:

1. The users will be able to discover and retrieve distributed services that are registered on regional or national SDIs and select them as the inputs of the vulnerability processes.
2. Status report message, probably in the form of a progress bar will be sent to the user while the processes are running.
3. The users will be able to stop the process if it takes too long to complete.
4. There will be a new section on the tool where the user can translate the vulnerability raster maps into classified road lines based on the vulnerability of the surrounded area.

1.2 OBJECTIVES & RESEARCH QUESTIONS

The main objective of this MSc thesis research is to try to create a new version of the RI2DE tool, under the concept of an free and open source OGC Web Services Architecture. In order to reach to such an objective, a thorough research should be made on the efficiency of the latest versions and the *state-of-the-art* implementations of the OWS to support the development of a tool with theses requirements, and on the appropriate design of an interface that offers a good user experience. At the end of this research the tool will have a new OGC Web Services architecture and a different GUI and workflow of the user actions. Therefore the main research question of this research is:

"What will be the new OGC Web Services Architecture of the tool", and is followed by a set of five sub-questions, that helps to achieve the main objective of this research.

1. *How flexible are the Web Processing Services of the tool?*

   The first sub-question aims to answer to the question if the susceptibility factor Web Processing Services of the tool are able to accept different inputs from the default ones. The first new requirement of the tool that concerns the discovery, retrieval and selection of the source service for the susceptibility factor processes, requires flexible WPS.

2. *How to get the metadata of the Web Services from the distributed Catalogues?*

   Governmental and local organizations and institutions that advertise the openness of the datasets, tend to publish them as OWS and register them as records with attached metadata in their SDIs (geoportals). The aim of the second sub-question is to find the way to retrieve the metadata (*OWS URL* of the server that they are published and *layer name* of the services that concerns their location.

3. *How to establish the connection between the Catalogue Services and Web Processing Service ?*

   The core of the discover, retrieve, display, select and use the sources as inputs of the susceptibility factors processes requirements is to succeed the coexistence of the CSW and the WPS in the same architecture schema. The aim of this sub-question is to define how these standards are going to be "connected", in order to pass from the metadata to the data itself, and to input them to the Web Processing Services.

4. *What is the status of the PyWPS 4.0.0 with respect to the Web Processing Services 2.0.2 version (test of asynchronous job control and job monitor)?*

   The free and open-source implementation of the Web Processing Service, *PyWPS*, has recently released the *PyWPS 4.0.0* version, which has been developed in order to implement the WPS 2.0.2 version of the standard. Since the *PyWPS* is a volunteer community the aim of this sub-question is to review this new version of the *PyWPS* regarding its status with the respect to the new operations of the standard (*GetStatus, GetResult, Dismiss*).

5. *What will be the GUI and work flow of the user actions?*

   This sub-question concerns the design of a new GUI and a new sequence of actions that a user of the tool can take, in order to provide the new functionalities, achieving high level user experience.

1.3 RESEARCH SCOPE

The main scope of this research will be the evaluation of the existing OGC Services and technology framework, with final goal to develop an efficient and self-descriptive new version of the RI2DE GIS web tool. The tool was the motivation to assess how the standards, the implementations and the organization themselves can support these new functionalities. So by trying to implement the functionalities that were set at the Motivation section this thesis will:

- Research and implement the WPS with the PyWPS 4.0.0, in order to test implementation in asynchronous execution and the state of it regarding the *GetStatus*, *GetResult*, *Dismiss* operations of the WPS
- Assess the flexibility of the WPS of the tool on accepting other services apart the default ones
- Assess the CSW standard and how much the organizations implement it by publishing their datasets, tools and services

- Find a solution for distributing search in the RI2DE platfrom through the CSW standard
- Design an as much as possible, user friendly and self descriptive new GUI for the tool

In the practical phase of this project, I will work both on the *back-end* and the *front-end* on the tool, trying to implement all the requirements. The tool will be evaluated from the developers, TU Delft mentors, stakeholders, in order to assess its usability and functionality. This MSc thesis project is happening in collaboration with TU Delft and Deltares institution, and together with this research, another group of developers of the OpenEarth initiative of Deltares, works on some extra functionalities of that the new version of the tool will provide.

1.4 METHODOLOGY

The objective of this section is to outline the general methodology that is going to be followed in order to conduct this research, aiming to answer all the research questions and achieve the scope of this project. This research project applies to the information systems field, and the methodology steps it was selected to follow the Design Science Research (DSR) approach as it was presented by Hevner and Chatterjee [2010] DSR approach aims at designing artifacts, such as tools, methods, and techniques that make information systems more effective and efficient. Following a problem solving approach DSR results in innovative artifacts tools, based on the state-of-the-art of the relative application context.

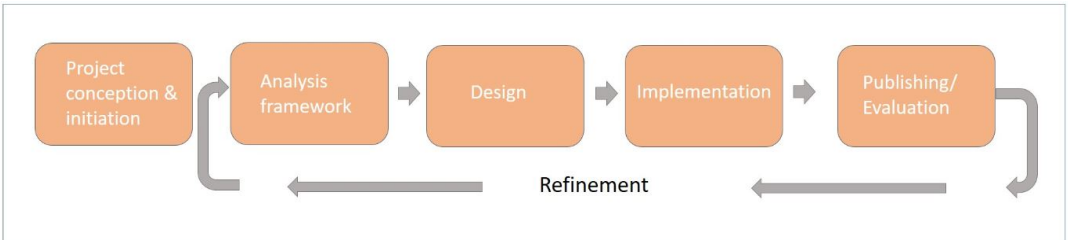

Figure 1.2: Methodology steps

The complete step-by-step process that is going to be followed is depicted in the Figure 1.2 and Figure 1.3 will outline how the artifact is going to be designed and how the relative research is going to be conducted. This process is consisted of a number of sequential phases, namely, Project conception and initiation, Analysis framework, Design, Implementation, and Publishing and Evaluation. Due to the multi-disciplinary nature of web design the whole procedure is iterative, which means that a phase can be refined multiple times before the delivery of the final research product. A detailed description of each phase will be given in the following sections.

1.4.1 Project initiation

This MSc thesis started when the supervisors/ stakeholders from Deltares institution, provided the first version of the tool for research purposes, and the first meeting with the mentors from TU Delft, took place. After several discussions on the gaps of the first version, the possible improvements on it, and the state-of-the art publications and implementations of the OGC Web Services, the research topic was defined. At the end of the first phase, the research questions and scope of the project, together with the new functionalities were set.

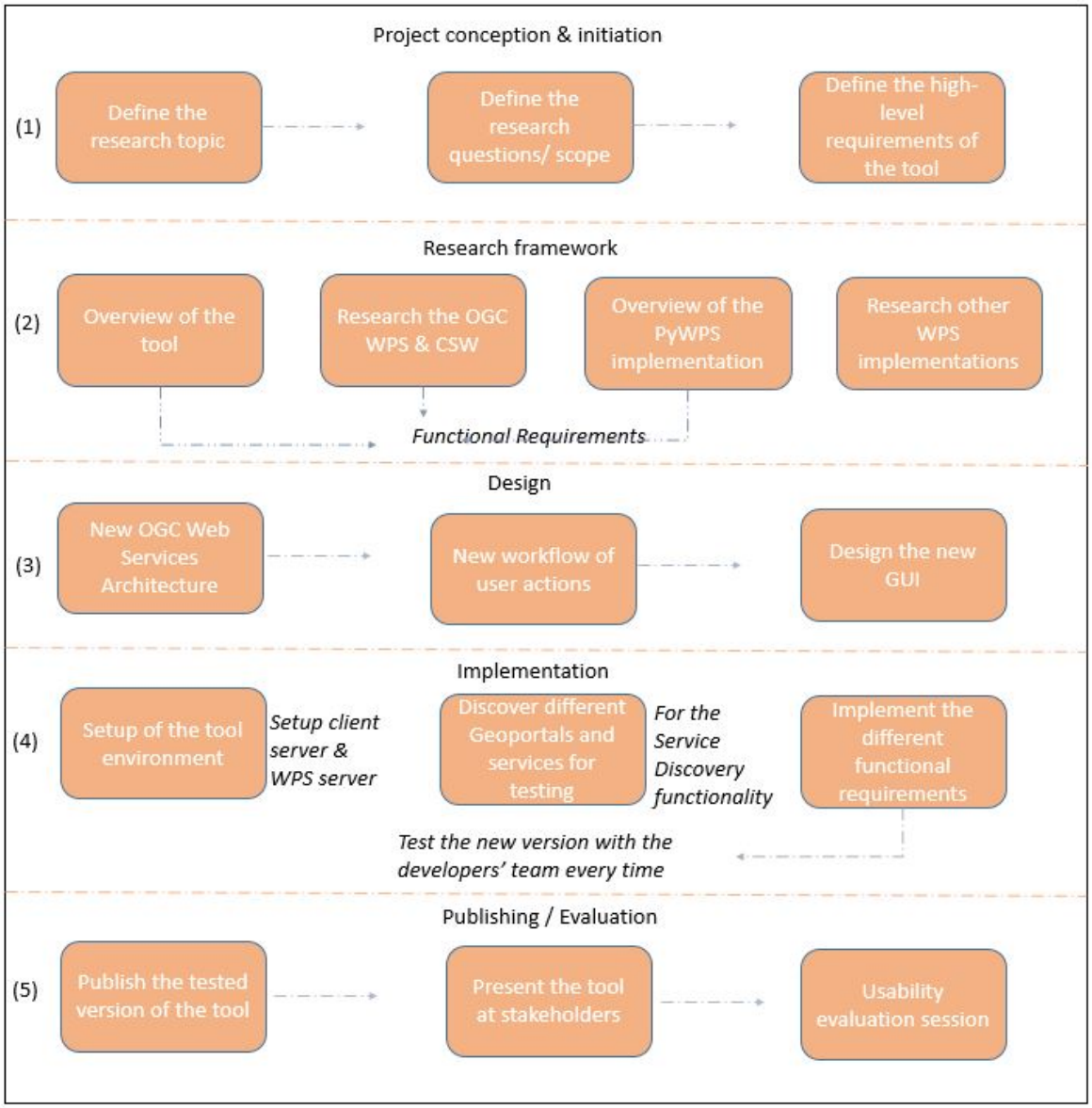

Figure 1.3: Analysis of methodology steps

1.4.2 Analysis framework

The research framework phase is the most important part of this research project, where a thorough research on the tool, the OGC Web Services and the state-of-the-art technology is being carried out. The result of this research will set the base for answering the sub-research question 2, 3, 4 and 5, and will examine the feasibility to implement the new functionalities under the existed technology framework. More specifically in this phase it has been carried out:

- A detailed overview of the first version of the tool, in order to review its' current architecture, the services that comprise, their flexibility, and the technology that it uses.
- A thorough research concerning the different OGC Web Services that can become components of a publish-find-bind OGC Web Service Architecture, focusing on the Catalogue Service for the Web communication protocol and the new operation of the Web Processing Service 2.0.2 version, *GetStatus, GetResult, Dismiss.*
- An examination on the new version of the free and open-source implementation of the Web Processing Service, *PyWPS 4.0.0*. The *PyWPS* has been mainly analyzed and tested in order to assess its' status with respects to the new operations *GetStatus, GetResult, Dismiss* of the *WPS 2.0.2*.
- A research on other WPS implementations in order to create a benchmark for the PyWPS implementation and to assess what is the state-of-the-art regarding the WPS 2.0.2 standard.

  The final product of the research framework phase is a functional requirements list for every new functionality.

1.4.3 Design

The design phase is the "skeleton" of this research project, as it describes how the new version of the tool is going to be, based on the functional and the high-level requirements. Together with the supervisors and stakeholders from Deltares, the mentors from TU Delft, and the developer's team it has been designed:

- The new OGC Web Services Architecture of the tool
- The new GUI of the tool.
- The new workflow of the user actions

The designed phase products are refined multiple times, according to the feasibility to have the requirements and the functionality/ usability of the tool. The final architecture of the tool is also the answer to the main research question of this project and is presented as a result in the implementation chapter (Section 6.6).

1.4.4 Implementation

The implementation phase comprise the practical part of this project, and involves the developing part of the tool, where the back-end and the front-end of the tool are constructed based on the designed OGC Web Services architecture and on the desired GUI of the tool. The implementation phase apart from the coding part, involves also the tests of the different versions of the tool from the developer team, regarding the functionality. In more details the implementation phase is comprised by the following steps:

- Firstly the RI2DE environment has been set up for development purposes. Both the back-end and the front-end of the tool has been set up and run at my personal laptop. At the end of this step, there is ready the environment for the development of the new services, and interface of the tool.
- The next step of the implementation phase has been the discovery of several OGC WCS that are registered in distributed geoportals, in order to be used for testing purposes for the functionality, disover, retrieve, display, select and use other services. The services should be appropriate inputs for the susceptibility factor WPS, and to be registered with all the necessary information for their retrieval and use.
- The most important step of the implementation phase is the integrate of the new functionalities, one by one according to the feasibility to implement them within the state-of-the-art technology framework. For an organized implementation procedure, the development is being carried out by trying to implement the functional requirements as they are described in the Chapter 5 chapter. Every time one functionality is implemented the tool is tested by the developers and then refined if needed.

1.4.5 Publishing & Evaluation

The last step of this research project, involves the publishing and evaluation of the tool. The publishing of the tool will be made each time a new tested version of it (tested from the developers) is also assessed from the stakeholders during a presentation, where the new functionalities are presented and new ideas and comments on the functionalities and usability of the interface are proposed. After the publishing of the tool an evaluation session should be carried out, in order to assess the workflow of the user actions, and how easy is for someone who is not part of the stakeholders and developers team, to understand the whole concept of the tool. Due to time constraints the usability session of the tool will future work.

1.5 THESIS OUTLINE

The report of this thesis is organized as follows: Chapter 2 carries out a research on the different OGC standars that comprise the architecture of the tool, and more thorough one on the WPS 2.0.2 and the CSW standards. Chapter 3 present the technology framework that is behind the implementation of the back-end and on the front-end of the tool, focusing on the PyWPS 4.0.0 implementation of the WPS standard. Chapter 4 chapters explores different WPS implementations in order to create a benchmark for PyWPS, a GIS project that publish its processes with the WPS 2.0 standard, the OpenEarth initiative of Deltares and the gives an overview of the **ri2d!** (ri2d!) tool. Chapter 5 describes the requirements that need to be fulfilled in order to have the new desired functionalities, in an OGC Web Services Architecture. Chapter 6 present the different steps that were followed for the development of the new functionalities, the new GUI and the new architecture. Chapter 7 concludes this research by answering to the research questions, outlining the issues, limitations and the achievements and giving some ideas for future work.

2 OGC WEB SERVICES AND OTHER STANDARDS

Following the broader context of web wervices and standardization, the OGC sets the rules for publishing geo-datasets and tools as web services with standard request and response interfaces. Within OGC general guidelines, an OWS oriented architecture is promoted for developing GIS web tools in an interoperable and open environment.

For every OWS service, OGC specifies unique rules and standard operations that the services may be called to execute. In order to request the execution of one of these operations, the HTTP communication protocol is used. More specifically, there are two ways to request an operation of an OWS:

- The GET method: Key-Value Pair (KVP) is used to encode the operation requests
- The POST method: XML encoding is used for operation requests or contained in the body of a Simple Object Access Protocol (SOAP) message

OGC has alliance with other international organizations that provides standards, such as ISO. At this section of the report are going to be presented the different standard file formats and the OWS that compose the OGC Web Services Architecture of the RI2DE tool. More specifically at Section 2.1 and Section 2.2 will describe the JSON and GeoJSON data formats respectively. Section 2.3 describes the Web Map Service that deals with the visualization of the datasets. Section 2.4 describes the Web Coverage Services that deals coverages. Section 2.5 describe the different version and operation of the Web Processing Service that deals with GIS processes. Section 2.6 describes the Catalogue Services specification for the development of catalogues. Section 2.7 describes the CSW binding protocol. Section 2.8 describes the ISO and Dublin core metadata schemas.

2.1 JSON

JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute-value pairs and array data types. JSON was derived from JavaScript and is language independent data format. While JSON is a data format, it is being used frequently as a configuration file Wikipedia [2019e].

2.2 GEOJSON

GeoJSON is JSON based open standard format designed for representing simple geographical features, along with their non spatial attributes. The features include points, lines, polygons and multipart collections Wikipedia [2019a].

2.3 OGC WEB MAP SERVICE (WMS)

The WMS specification provides guidelines for the publishing of geo referenced maps for visualization purposes. A map is not the data itself. WMS produced

maps are generally rendered in a pictorial format such as *PNG*, *GIF* or *JPEG*, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats Beaujardiere [2006]. The OGC released the first version of the WMS in 2000, and the latest release is the WMS 1.3.0 in 2006.

The Web Map Service defines three operations:

- **GetCapabilities** (Mandatory): The purpose of GetCapapilities operation is to retrieve the service metadata.
- **GetMap** (Mandatory): The purpose of a Get Map operation is to returns a map whose geographic and dimensional parameters are well-defined.
- **GetFeatureInfo** (Optional) : The GetFeature operation returns information about particular features shown on a map. It is only supported for those layers for which the attribute queryable equals to true.

2.4 OGC WEB COVERAGE SERVICE (WCS)

The WCS defines standard interfaces for the publication of coverages (digital geospatial information representing space/time-varying phenomena) as web services. Unlike the WMS, which deals with static maps for only visualization purposes (just pictures), WCS deals in theory with spatio-temporal regular and irregular grids, point clouds, and general meshes, but in practice only grids, and is possible to get the original data for further processing Baumann [2010]. The latest release of the standard is the WCS 2.1 in 2018.

- **GetCapabilities** : The purpose of GetCapabilities is to retrieve the service metadata and coverages offered by a WCS server.
- **DescribeCoverage**: The purpose of this operation is to retrieve the description of specific coverages that are included in the request.
- **GetCoverage**: The GetCoverage operation allows a client to request the coverage itself or a part of it.

2.5 OGC WEB PROCESSING SERVICE (WPS)

WPS is the OGC standard, that defines how geo-spatial processes, algorithms, and calculations can be published and invoked over the web. WPS provides rules for standardizing requests and responses (inputs and outputs), between a client and a WPS server. The input datasets to a Web Processing Service can be delivered either across a network or they can be available at the same server as the processing service. Since the first WPS specification published from OGC, several changes and improvements have been made. Currently the WPS 2.0.2 is the latest version of the standard and it has been modified from the previous one (WPS 1.0.0) with added functionalities (job control and job monitoring) two important operations for the complex and long time processes.

2.5.1 OGC WPS 1.0.0

- **GetCapabilities:** The GetCapabilities operation, request for an XML response that provides the WPS server properties. In these properties are included the names of the Web Processing Services that the server offers, and general description concerning them.

- **The DescribeProcess:** The DescribeProcess operation, request and receives information, in an XML form, concerning the description of one or more Web Processing Services that the server offers, and are executable.
- **The Execute:** The Execute operation allows WPS clients to trigger the execution of a specific Web Processing Service. The inputs to the process can be either included directly in the Execute request, or reference web accessible resources. The result to the Execute request is either embedded in the response XML document or stored in as accessible resource, and the URL of the stored output in referred in the response document.

In order to invoke the execution of a Web Processing Service according to the 1.0.0 version of the standard, inside the Execute request should be given the following properties Schut [2007].

- Service type identifier
- Operation name
- Version of the service
- Identifier of the process
- List of inputs provided to this process execution Data type: Complex data, Literal Data, Bounding Box data
- Response form of the result (It can be set either to "Raw Data output" which indicates that the output shall be returned directly as raw data, without a WPS response document, or "Response Document" which indicates that the output shall be returned as part of a WPS response document
- Language

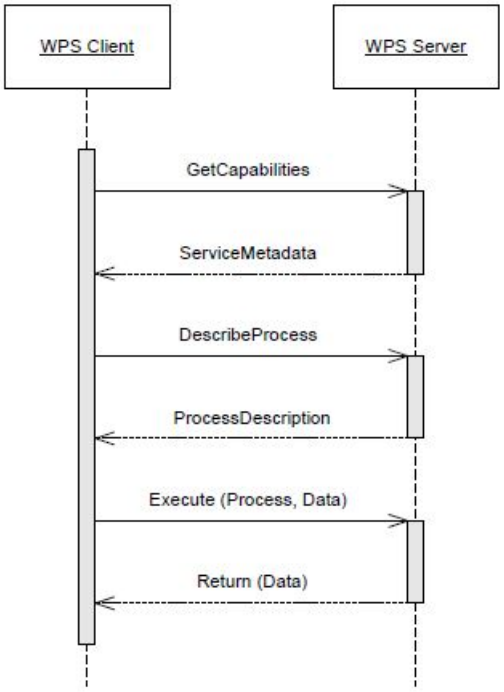

Figure 2.1: Mandatory operations of WPS in synchronous communication

While the Job monitoring (status reporting of the process) was formally introduced as an operation from the WPS 2.0.2 version of the standard, there was already

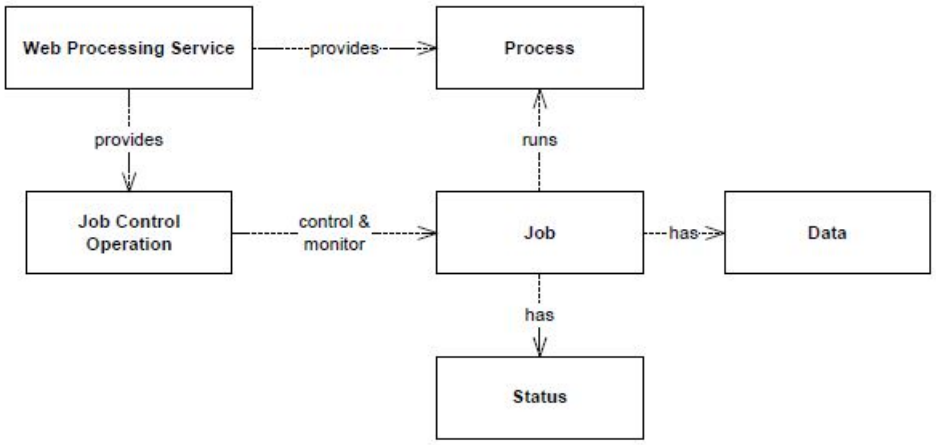

Figure 2.2: WPS 2.0.2 Conceptual Model, Source: OGC WPS 2.0.2 Interface Standard: Corrgendum 2

a way to have ongoing reporting of the processing from WPS 1.0.0, by setting the Response Form property of the Execute response to "Response Document" and the "store" and "status" elements of the property to "true".

More specifically setting the "store" element to true, indicates that the Execute response document shall be stored at a web accessible URL (third party server), and if the "status" element is true, this document shall be updated while the process is running. In this document the "status" element can have the following values Schut [2007] :

- "Process Accepted" : The first time the document is returned
- "Process Started": While the process is running this message is being send
- "Process Paused"
- "Process Finished" : When the process finish without a problem
- "Process Failed" : If the process execution fails for some reason

2.5.2 OGC WPS 2.0.2

The WPS 2.0.2 version of the standard has been released in 2015, almost 10 years later from the previous version. In addition to the mandatory operations that the WPS 1.0.0 version of the standard offers, the WPS 2.0.2 version introduced three new operations and asynchronous execution of the process, in order to provide job monitoring and job controlling. The WPS conceptual model of the 2.0.2 version of the standard, as it is illustrated in the Figure 2.2, defines that every time a process runs on the server, a job is created with unique id. This job has a result, has a status and can be controlled by the service. Having this model as baseline the WPS 2.0.2 version of the standard, extra to the three mandatory operations, GetCapabilities, DescribeProcess and Execute, offers three new optional operations:

- **GetStatus**: This operation allows a client to query status information of a processing job (Can be invoked only when the process runs on asynchronous mode)
- **GetResult**: This operation allows a client to query the results of a processing job (Can be invoked only when the process runs on asynchronous mode)
- **Dismiss** This operation offers control of the job in an asynchronous mode

Table 2.1: Possible values of status element in Status Info Document

Status	Definition
”Accepted”	The job is queued for execution
”Running”	The job is running
”Succeeded”	The job has finished with no errors
”Failed”	The job has finished with errors
”Dismissed”	The job has been canceled

As it is outlined above, the GetResult and the GetStatus operation can be invoked only when the process runs on asynchronous mode, which is a new functionality of the WPS that was introduced from the 2.0.2 version of the standard. More specifically in the Execute operation properties of the WPS 2.0.2 there is a property that indicates if the process will run on synchronous or on asynchronous mode (See below a list with the Execute operation properties of WPS 2.0.2).

- Service
- Version
- Extension
- Response format: Result Document, StatusInfo Document, Raw data
- Execution mode: synchronous or asynchronous
- Input
- Output

In the synchronous case, which is suggested when the processes don't demand too much time to complete, the client submits an execute request to the WPS server and the connection between them stays ”open” until the processing job finish (Figure 2.3) Mueller and Pross [2016]. In the asynchronous case the client send an execute request to the WPS server and immediately receives a status information response (the Status Info document) which indicates that the request has been accepted and that a processing job has been created and will be run in the future, then the connection ”closes”. In the StatusInfo document there is also information regarding the Id of the job and the URL location. Now that the process runs on asynchronous mode there is the option to monitor the job by sending GetStatus requests using the job Id. The response to a GetStatus request is a StatusInfo document which contains:

- JobID
- Status: See Table 2.1
- Expiration Data of the job
- An estimation of the completion
- Next poll
- Percentage Completed

When the process finishes, the client can send a GetResult request, in order to receive a Result Document with the output of the process.

Moreover to the job monitoring, as it is already mentioned, the WPS 2.0.2 version of the standard offers also the option to control the job, using the Dismiss operation. The Dismiss operation, request from the WPS server to erase the Job Id of the process, even if the process is still running, which means that the execution is cancelled. The result to Dismiss request is StatusInfo document with status ”Dismissed”. The Dismiss operation is usable for long-time running processes and in order to release server resources Mueller and Pross [2016].

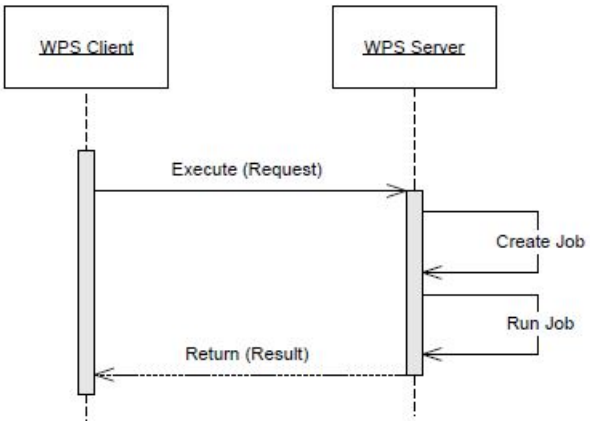

Figure 2.3: Synchronous process execution, Source: OGC WPS 2.0.2 Interface Standard: Corrgendum 2

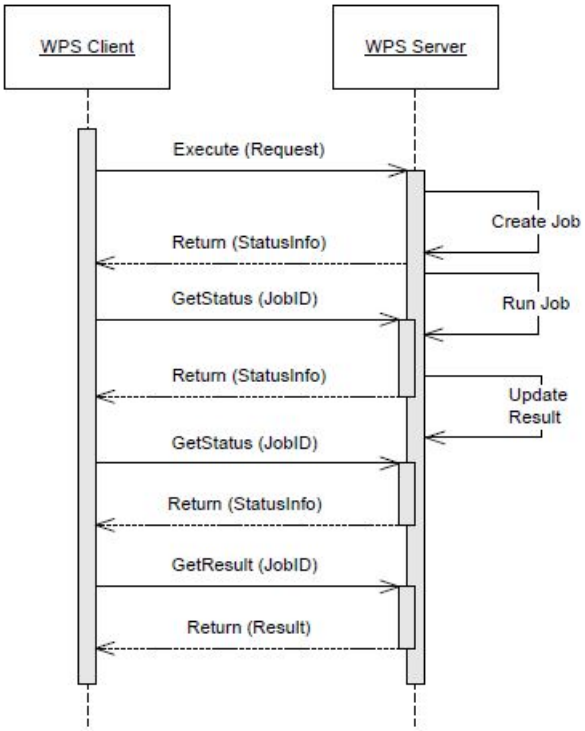

Figure 2.4: Asynchronous process execution, Source: OGC WPS 2.0.2 Interface Standard: Corrgendum 2

2.6 OGC CATALOGUE SERVICES

One of the most essential part of an SDI is its catalogue server. The catalogue of the geo-portal, is like an online library of geo-datasets and processes, where the user can discover them through their descriptive characteristics e.g. data of creation, title, geographic area. These descriptions of geospatial data are called metadata ("data about data"). In order to promote the sharing of geoghraphic information throughout the maximum number of users, it is necessary to create distributed networks of catalogs that are developed under common rules, and that use a standardized mechanism for catalog querying Nogueras-Iso et al. [2005]. Web catalogues like that could improve geospatial information sharing between organizations and their users, using the capacities of the Internet, reduce duplication, and enhance information consistency and data preparedness.

On that spirit the Open Geospatial Consortium published the OGC Catalogues Services specification that provides rules and guidelines for the development and publishing of Catalogues Services, that have standard interfaces for the discovery, accessibility, maintenance and organization. In a more formal definition, OGC Catalogue Services support the ability to "publish" and "search" collections of descriptive information (metadata) for data, services, and related information objects Nebert et al. [2016b]. OGC Catalogue Services guidelines have evolved and improved through years. The latest releases of the standard are the OGC Catalogue Services 2.0.2 (released in 2007) and the OGC Catalogue Services 3.0 (released in 2016). Since the difference between these specifications are minor for the purpose of this research, the guidelines of the OGC Catalogue Services 3.0 are going to be used in order to describe the service.

In OGC Catalogue Services specification it is defined a general architecture that specifies an abstract information and interface model that every catalogue should have. The information model involves Nebert et al. [2007]:

- **A minimal query language**

  The interoperability goal is supported by the specification of a minimal abstract query (predicate) language that is called *OGC_Common Catalogue Query Language*. This minimal query language shall be supported by all compliant OGC Catalogue Services and should assist the consumer in the discovery of datasets of interest at all sites that support the OGC Catalogue Services specification. *OGC_Common Catalogue Query Language* supports Boolean queries, text matching operations, temporal data types, and geospatial operators and its' language syntax is based on the SQL WHERE clause in the SQL SELECT statement Nebert et al. [2007].

- **A set of queryable attributes (names, definitions, conceptual datatypes)**

  The goal of defining core queryable properties is to offer query interoperability among catalogues, which means that the same queries can be executed against any catalogue service without modification and without detailed knowledge of the catalogue's information model. Table 2.2 defines the set of the abstract queryables Nebert et al. [2007].

- **A set of core returnable properties**

  A set of core properties returned from a metadata search i encouraged to permit the minimal implementation of a catalogue service. The set of metadata that is going to be returned is defined from the chosen group (the Common Element Set). There are three groups: "brief", "summary", and "full". The returnable properties are expressed using the syntax of Dublin Core metadata elements (Figure 2.5).

- **A common record format**

  A common record format is defined in the OGC Catalogue Services specification, that outlines the minimal set of elements that should be returned in the brief and summary element sets Nebert et al. [2007].

The general interface model provides a set of abstract interfaces (classes) that support the discovery, access, maintenance and organization of the catalogues. OGC Catalogue Services specifies for every interface, operations that provide functionality on a specific area of interest (e.g. discover the datasets). More specifically the general Catalogue Service class as it is described by the Catalogue Service 3.0 version of the specification can be associated with the:

- **OGC_Service class**: Provides the *getCapabilites* operation to retrieve the Catalogue service metadata and the getResourceById operation that will retrieve an object by query on its identifier only.

Dublin Core element name	Term used in OGC queryables	Definition	Data type
title	Title	A name given to the resource. Also known as "Name".	CharacterString
creator		An entity primarily responsible for making the content of the resource.	CharacterString
subject	Subject	A topic of the content of the resource. This is a place where a Topic Category or other taxonomy could be applied.	CharacterString
description	Abstract	An account of the content of the resource. This is also known as the "Abstract" in other aspects of OGC, FGDC, and ISO metadata.	CharacterString
publisher		An entity responsible for making the resource available. This would equate to the Distributor in ISO and FGDC metadata.	CharacterString
contributor		An entity responsible for making contributions to the content of the resource.	CharacterString
date	Modified	The date of a creation or update event of the catalogue record.	ISO-8601 date
type	Type	The nature or genre of the content of the resource.	CodeList
format	Format	The physical or digital manifestation of the resource.	CharacterString
identifier	Identifier	A unique reference to the record within the catalogue.	Identifier
source	Source	A reference to a resource from which the present resource is derived.	CharacterString
language		A language of the intellectual content of the catalogue record.	CharacterString
relation	Association	The name of the relationship that exists between the resource described by this record and a related resource referenced using the *Source* or *dc:source* property.	CodeList
coverage	BoundingBox	The spatial extent or scope of the content of the resource.	
rights		Information about rights held in and over the resource.	CharacterString

Figure 2.5: Common returnable properties, Source: OpenGIS Catalogue Services Specification 2.0.2

Table 2.2: Common queryable elements, Source: OpenGIS Catalogue Services Specification 2.0.2

Name	Definition
Subject	The topic of the content of the resource
Title	A name given to the resource
Abstract	A summary of the content of the resource
AnyText	A target for full-text search of character data types in a catalogue
Format	The physical or digital manifestation of the resource
Identifier	An unique reference to the record within the catalogue
Modified	Date on which the record was created or updated within the catalogue
Type	The nature or genre of the content of the resource. Type can include general categories, genres or aggregation levels of content.
BoundingBox	A bounding box for identifying a geographic area of interest
CRS	Geographic Coordinate Reference System for the Bounding Box
Association	Complete statement of a one to one relationship

- **Discovery class:** Provides the *query* operation for searching the catalogues metadata and retrieve a set of all the resources that satisfy the query, the *describeRecordType* operation retrieves the type definition used by metadata of one or more registered resource types, the *getDomain* operation retrieves information about the valid values of one or more named metadata properties.

- **Manager class:** Provides the *transaction* operation that performs a specified set of ”insert”, ”update”, and ”delete” actions of metadata items stored by a Catalogue Service implementation and the *harverstResrouce* operation that requests the Catalogue Service to retrieve resource metadata from a specified location.

While OGC Catalogue Services specifies a general architecture for the information and the interface model that every digital catalogue should provide, it does not require the use of a specific metadata schema. Where a catalogue service advertises a specific application schemas, catalogues that handle geographic dataset descriptions should conform to published metadata standards and encodings of this schema.

The abstract information and interface model that the OGC Catalogue Services defines, is implemented by protocol bindings. Each protocol binding includes a mapping from the general interfaces, operations and parameters that were specified in this section, to the constructs available in the chosen protocol. For the purpose of this research project the HTTP (OGC Catalogue Services for the Web) protocol binding has been research thorough (see Section 2.7).

2.7 OGC CATALOGUE SERVICES FOR THE WEB (CSW)

At this section the CSW protocol binding is going to be presented. Having as baseline the HTTP protocol binding the CSW sends the requests to the catalogue and receives the responses, using the GET or POST methods. This standard builds on the general model of the Catalogue Services and offers seven operations: *GetCapabilities*, *GetRecordsById*, *GetRecords*, *GetDomain*, *Transaction*, *Harvest and Unharvest*. . In the Table 2.3 the CSW operations are mapped to the Catalogue Services operations.

Table 2.3: General model to CSW mapping

General Model Operation	CSW Operation
OGC_Service.getCapabilities	GetCapabilities
OGC_Service.GetResourceById	GetRecordById
Discovery.query	GetRecords
Discovery.getDomain	GetDomain
Manager.transaction	Transaction
Manager.harvestRecords	Harvest
Manager.harvestRecords	UnHarvest

2.7.1 Query predicate language at CSW

As it was already mentioned in the OGC Catalogue Services section, the abstract information model indicates the need for a common predicate language but it does not specifies the encoding to be used. The OGC CSW defines two predicate languages Nebert et al. [2016a]:

- Common Query Language (CQL)_TEXT: Is a text encoding of the Backus Naur Form (BNF).
- FILTER: Is an XML encoding of the BNF grammar and is defined in the Filter Encoding Implementation Specification Vretanos [2005].

All CSW implementations that implement the XML filter encoding syntax as defined in the OGC Filter Encoding Implementation Specification, shall support at least the following operators Vretanos [2010]:

- Logical operators: (And, Or, Not)
- Comparison Operators: (PropertyIsEqualTo, PropertyIsNotEqualTo, PropertyIsLessThan, PropertyIsGreaterThan, PropertyIsLessThanOrEqualTo, PropertyIsGreaterThanOrEqualTo, PropertyIsLike, PropertyIsBetween
- Spatial operators: BBOX
- Temporal operators: TOverlaps

2.7.2 Core queryable & returnable realization

The CSW protocol binding, specifies the csw:AbstractRecord for the realization of the abstract record that the general information model demands. The csw:AbstractRecord represents all the three groups (”brief”: csw:BriefRecord, ”summary”:csw:SummaryRecord, ”full”:csw:SummaryRecord) of the Common Element Set and its’ syntax is an XML-based encoding based on the *Dublin Core* metadata (see Section 2.8.2). Figure 2.6 maps the *Dublin Core* element names of the OGC query and response elements specified in general model, to the concrete XML elements specified in this standard.

2.7.3 GetRecords operation

The GetRecords operation of the CSW binding protocol, maps to the Discover.query operation of the general model. The GetRecords request implements a search operation in the Catalogue, based on a query, and retrieves the resources that fulfills the query. In the Table 2.4, the a part of the most important to this research input parameters of a GetRecords request are presented.

The response to a GetRecords request is an XML that contains the result of the search operation, on the selected XML schema. An overview of the elements that are contained in the GetRecords result response, is offered in the Table 2.5.

Dublin Core element name	OGC queryable[2] term (abstract)	XML element name[1] (to be used in query and response)	Datatype
title	Title	dc:title	character string
creator		dc:creator	character string
subject	Subject	dc:subject	character string
description	Abstract	dct:abstract	character string
publisher		dc:publisher	character string
contributor		dc:contributor	character string
date	Modified	dct:modified	date
type	Type	dc:type	character string, from a predefined set
format	Format	dc:format	character string
identifier	Identifier	dc:identifier	character string, an identifier
source	Source	dc:source	character string
language		dc:language	character string, from a predefined set (see IETF RFC 4646)
relation	Association	dc:relation	
rights		dc:rights	
coverage	BoundingBox	ows:BoundingBox	gml:Envelope
coverage	TemporalExtent	csw: TemporalExtent	gml:TimePeriod
	AnyText	csw:AnyText	character string

1. These XML elements are contained as children of the csw:Record element.
2. A blank value indicates that there is no corresponding OGC queryable term.

Figure 2.6: Mapping of Dublin Core names to XML element names, Source: OpenGIS Catalogue Services Specification 3.0

Table 2.4: Part of the Parameters fot GetRecords operation request

Parameter	Description
REQUEST	Name of the operation: GetRecords (Mandatory)
Service	Name of the service: CSW (Mandatory)
Version	Version of the standard (Mandatory)
NAMESPACE	Prefixes of the elements (Optional)
outputFormat	Default value is xml (Optional)
outputSchema	Specifies the XML schema - default value is http://www.opengis.net/cat/csw/3.0 (Optional)
maxRecords	The maximum number of records to be returned from the search
typeNames	The typenames to query against, e.g csw:Record (Optional)
ElementSetName	”full’, ”summary” or ”brief” (Optional)
CONSTRAINTLANGUAGE	CQL_TEXT or FILTER (Optional)
Constraint	The query (Optional)

Table 2.5: Search result parameters in GetRecords response

Result Parameters	Description
result id	A server generated identifier for the result.
element set	The element set that have been returned: ”brief”, ”summary”, ”full”
record schema	A reference to the type or schema of the records returned
number of records matched	Number of the records found by the GetRecords operation
number of records reuterned	Number of records actually returned to client
next record	Start position of next record
expires	Indicates when the result will expire

2.8 METADATA SCHEMES

In order to be useful, metadata needs to be standardized. This includes agreeing on natural language, spelling, date format, etc. If everyone uses a different standard, it can be very difficult to compare data to other data UNC [2019].

A key component of metadata is the schema. Metadata schemes are the overall structure for the metadata. It describes how the metadata is set up, and usually addresses standards for common components of metadata like dates, names, and places. There are also discipline-specific schemes used to address specific elements needed by a discipline UNC [2019]. For the purpose of this project, two metadata schemes have been researched, that are usually used for the description of the records in the different catalogue services. At this section is given a small description of the ISO and Dublin Core metadata schemes. .

2.8.1 ISO metadata schema

ISO is the International Organization for Standardization. They develop and publish International Standards. ISO [2019]. ISO creates documents that provide requirements, specifications, guidelines or characteristics that can be used consistently to ensure that materials, products, processes and services are fit to the general purpose of standardization ISO [2019]. The geomatics standardization activity in ISO Technical Committee 211 include formal schemes for geospatial metadata that are intended to apply to all types of information.

These metadata standards, ISO 19115:2003 and ISO 19115-1:2014 include proposals for core (discovery) metadata elements in common use in the geospatial community. ISO/TC 19139:2007 defines a formal encoding and structure of ISO 19115:2003 metadata for exchange.

2.8.2 Dublin Core

Dublin Core is a general standard first used by libraries, and can be adapted for specific disciplines. The Dublin Core Metadata Element Set is one of the simplest and most widely used metadata schema. Originally developed to describe web resources, Dublin Core has been used to describe a variety of physical and digital resources. Dublin Core is comprised of 15 “core” metadata elements, whereas the ”qualified” Dublin Core set includes additional metadata elements to provide for greater specificity and granularity UNC [2019]. :

3 TECHNOLOGY FRAMEWORK

This section will give an overview of the technology framework that the RI2DE GIS web tool use for the back-end implementation of the OWS and for the client-side programming (RI2DE web browser). The different technology components that the tool use are free and open source. More specifically Section 3.1 will present the *PyWPS* implementation of the WPS standard, focusing on the current situation, regarding the implementation of the *GetStatus*, *GetResult* and *Dismiss* operations of the version 2.0.2 of the standard, and its ability to perform *asynchronous execution* of the process. Section 3.2 will describe the *GeoNetwork* implementation for the creation of catalogues that support the CSW binding protocol. Section 3.3 will briefly present the *GeoServer* software that was used for publishing the raster datasets of the tool as WMS and WCS. Section 3.4 will give an overview of the *Vue.js* framework of *JavaScript* and its' state management library, *Vuex*, that were used for the development of the RI2DE web browser. Section 3.5 will describe the *OwsLib* client programming python library that was used for the implementation of the *GetRecords* request.

3.1 PYWPS

PyWPS is the server side implementation of the OGC WPS standard, written in the Python programming language PyWPS [2019b]. It was one of the first implementations of the OGC WPS and tries to connect to all existing tools for geospatial data analysis, available on the Python platform. *PyWPS* is taking care of security, data download, request acceptance, process running and final response construction.

In 2006 *PyWPS* 2.0.0 was released supporting the WPS 0.4.0 version of the standard, and as the standard evolves, *PyWPS* implementation adjust to the newest version of the protocol. *PyWPS- 3* series supported the WPS 1.0.0 version of the standard, and currently the *PyWPS- 4* version has been developed and released in order to support eventually the WPS 2.0.0 of the standard.

PyWPS is a free and open source solution for the implementation of the WPS standard, and its development depends mainly on volunteers work and on students project. For that reason it is essential at this part of the research to outline the current situation of *PyWPS*, what it offers, and how complete is the implementation of the *GetStatus*, *GetResult* and *Dismiss* operations.

3.1.1 PyWPS-4

PyWPS 4.3 is the most current version of the *PyWPS*, and it has been developed almost one decade after the previous one. The *PyWPS* community decided to rewrite

Figure 3.1: PyWPS logo

the *PyWPS* code from scratch, based on new knowledge and new technologies, having as main goal to fully implement eventually the WPS 2.0.0 version of the standard, and more specifically to support asynchronous execution and the *GetStatus*, *GetResult*, and *Dismiss* operations. The major changes in the *PyWPS* 4 are PyWPS [2019b]:

- *PyWPS* is written in Python 3 and runs on Python 2.7, 3.3 or higher.
- It utilizes native Python binding to existing projects (GRASS GIS)
- Supports GeoJSON, Keyhole Markup Language (KML) or TopoJSON
- *OWSLib* library is used for some data types
- *PyWPS* project has changed the licence from GNU/GPL to MIT
- *Lxml* library is used to handle XML files

PyWPS functionality

PyWPS does not come with any predefined processes. The processes has to be developed by the user. The essential functions of a *PyWPS* Čepický and de Sousa [2016] are:

- Provide the WPS communication bridge
- Read the input data that are in the Execute request
- Calling the Execute function of the process
- Provide progress reports
- Create and communicate the results to the client
- Run up to a determined number of processes in parallel
- Save a number of processes on queue for future execution

PyWPS processes and services

PyWPS works with processes and services. In *PyWPS* there is a service instance under which a collection of selected processes is published. A process is a Python class that contains a handler method and a list of input and outputs. *PyWPS* recognize all the defined inputs of WPS (*LiteraData*, *ComplexData*, *BoundingBox*). In order to code a *PyWPS* process, it needs to be created a new class that inherits from the *PyWPS* process class and needs the following attributes and functions in order to be configured:

- Identifier: Unique identifier of the process in order to be used in the execute request
- Title: Corresponding title of the process
- Abstract: A small abstract that defines the functionality of the process
- Metadata: Reference to more metadata about this process
- Inputs: A list of the inputs that needs to be contained in the Execute request
- Outputs: A list of the outputs of the process
- Store supported: Boolean (True for enabling the store of the output)
- Status supported: Boolean (True for enabling the status reporting functionality of the *PyWPS*)
- Handler method: The handler method accepts as inputs the request and the response objects of the process, and contains all the processing.

PyWPS configuration

The *PyWPS* instance comes with a configuration file that use the *ConfigParser* format. This configuration file contains the following sections PyWPS [2019b]:

- Metadata: contains the metadata of the service (like provider name and contact information)
- Server: contains information regarding the server (such as the maximum number of processes to run parallel or the output URL to store the results)
- Processing: configures the back end processing. Possible values: multiprocessing, scheduler and default which is the same as multiprocessing.
- Logging: for logging configuration
- Grass: for optional configuration to support *GRASS GIS*

PyWPS status reporting

Although *PyWPS* does not support the *GetStatus* and the *GetResult* operations yet, there is way for getting status report messages similar to the way that WPS 1.0.0 offers status report.

The status report messaging in the *PyWPS* can be enabled with the following steps:

1. Set the *store supported* attribute in the *PyWPS* process instance to "true"
2. Set the *status supported* attribute in the *PyWPS* process instance to "true"
3. Set process status in the handler method using the *WPSResponse.update_status* function.
4. Set the output URL location in the configuration file

By setting the status supported and the store supported to "true" and sending an execute request where the "ResponseFormat" parameter is equal to "ResponseDocument", the asynchronous request of the process is enabled. In the Listing 3.1 it is depicted the part of the Execute request that enables the asynchronous execution and in the Listing 3.2 it is presented, how the process status has been defined (using the *response.update_status* function, in a WPS that creates buffer of multiple features.

Listing 3.1: The part of the Execute request that enables asynchronous processing

```
1  ...
2  <wps:ResponseForm>
3      <wps:ResponseDocument>
4          <wps:Output asReference="true">
5              <ows:Identifier>output</ows:Identifier>
6              <ows:Title>Some Output</ows:Title>
7          </wps:Output>
8      </wps:ResponseDocument>
9  </wps:ResponseForm>
10 ...
```

The response document is stored in the output URL, and the client can have access to it with the *HTTP-GET* method. In this document the status is described either with PyWPS [2019b]:

- **ProcessAccepted:** Process was accepted by the server and the process execution will start soon
- **ProcessStarted:** Process calculation has started. The status also contains report about the percentage that is already done, and a status message (text reporting about the calculation state)

- **ProcessFinished:** Process instance performed the calculation successfully and the final Execute response is returned to the client or stored on final location
- **ProcessFailed**: There was something wrong with the instance and the server reports server exception.

Listing 3.2: Buffer multiple features process: Status report

```
def_handler(self,request,response):
    ...


    while index < featureCount:

        inFeature = inLayer.GetNextFeature()
        inGeometry = inFeature.GetGeometryRef()

        ...
        index += 1

        response.update_status('Buffering',
        100*(index/featureCount))
    ...
```

PyWPS Dismiss

PyWPS 4.3 does not support yet the Dismiss operation of the WPS 2.0.2 version of the standard. From the *PyWPS GitHub* repository it seems that there is an undergoing implementation by multiple *PyWPS* community members, with the most interest the master thesis of Laza [2018]. In this thesis the student tries to implement *Docker Container* isolation in *PyWPS* in order to isolate each process execution. The goal of this was to provide some mechanisms to control the execution of the process, which is essential for the *Dismiss* operation of the *WPS 2.0.2* standard. Several issues prevented the integration of the *Docker extension* into the official *PyWPS* repository. At the time of submitting this thesis project the *Docker Container* remains still on the wish list of the PyWPS community PyWPS [2019a].

3.2 GEONETWORK

GeoNetwork is an open source catalog application to manage spatially referenced resources. It is written in Java and its' latest stable release is the *GeoNetwork* 3.4.4. *GeoNetwork* is a standardized and decentralized system based on the concept of distributed data and information ownership, and is designed to enable access to geospatial datasets through descriptive metadata. *GeoNetwork* provides creating and editing of metadata and offers search functions as well as embedded interactive web map viewer. *GeoNetwork* offers support of OGC CSW 2.0.2 with the CSW operations, *GetCapabilities, DescribeRecord, GetRecordsById, GetRecords, Harvest, Transaction* GeoNetwork [2019].

Figure 3.2: GeoNetwork logo

3.3 GEOSERVER

GeoServer is an open source server written in Java that allows users to share, process and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards Wikipedia [2019b]. *GeoServer* implements OGC protocols such as WMS and WCS. Additional formats and publications are available as extensions including the WPS GeoServer [2019a].

Figure 3.3: GeoServer logo

3.4 VUE.JS AND VUEX

Vue.js or commonly *vue* is an open-source framework of *JavaScript* for building user interfaces of web applications. This framework offers an adoptable architecture that focuses on declarative rendering and component composition. For complex applications creation with *vue*, the *Vuex state management* library can be used Wikipedia [2019f]. The state management refers to the management of the state of one or more user interface controls such as text fields or radio buttons in a GUI Wikipedia [2019d].

Vuex library is a great solution for web applications that manipulate a big amount of datasets, that change all the time. It serves as a centralized store for all the components in an application, with rules ensuring that the *state* can only be mutated in a predictable fashion Vuex [2019]. A schematic representation of *Vuex* philosophy is depicted in the Figure 3.4. The *mutations* are the functions that are responsible for changing the values of the *state*, and the actions exist in order to commit the *mutations*. The *DevTools* is a set of web developer tools built directly in the web browsers in order to diagnose on-the-fly problems of the web application.

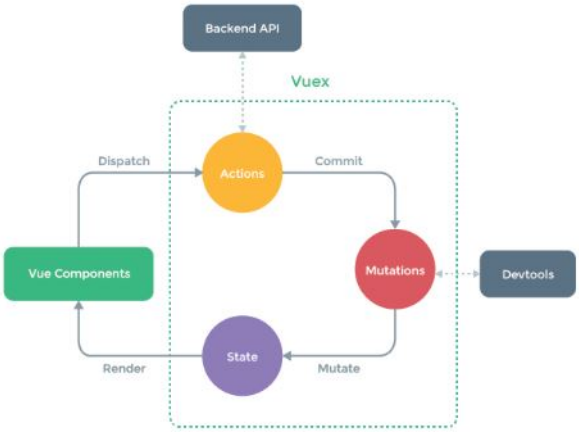

Figure 3.4: Schematic representation of Vuex philosophy, Source: Vuex [2019]

3.5 OWSLIB

OWSLib is a Python package for client programming with OWS. More simple *OWSLib* creates simple OWS clients in the form of python scripts. It implements the interfaces of the different operations that the standards offers, creating the XML requests, sending them and reading the responses. *OWSLib* has simple requirements. Needs only a python interpreter, as well as *ElementTree* or *lxml* python libraries for XML parsing Kralidis [2019] .

4 RELATED WORK

As it was already mentioned this thesis focus on researching the WPS standard, its free and open source python implementation *PyWPS*, and the CSW protocol for developing catalogues. In this chapter, at Section 4.1 several projects that implement the OGC WPS standard are going to be presented in order to have an overview of the different solutions that exist for the publishing of geoprocesses as WPS, and what is the state of them regarding the *version 2.0.2* of the standard. Then at Section 4.2 an example implementation of the WPS 2.0 standard with the ZOO-Project is given, from the e-government standard framework of South Korea. At Section 4.3 the OpenEarth initiative of Deltares is presented, and the MI-SAFE Viewer which has been an inspiration for the integration of the CSW at the RI2DE architecture. Finally at Section 4.4 a detailed overview of the architecture components and the technology behind the RI2DE tool.

4.1 WPS IMPLEMENTATIONS

The WPS is the standard that provides rules for standardizing requests and responses, for the publishing of geospatial processing. The most recent version of the standard is the WPS 2.0 that was released in 2015 and revised at the WPS 2.0.2 version in 2018. In the global geospatial community there are several projects that implement this standard in different ways and programming languages.

4.1.1 ArcGIS Server

ArcGIS Enterprise is a full-featured GIS system, powered by *ESRI*, offering the technology that makes possible to visualize, analyze and manage geospatial information. One of ArcGIS Enterprise products is the *ArcGIS server*, which makes possible to publish geospatial datasets and geoprocesses as services (either OWS or as other *ESRI* services ESRI [2019b]. With *ArcGIS server* it is possible to publish geoprocesses that were created within an *ArcGIS* environment (e.g. geospatial tasks that were developed with the *ArcPy* python package of ESRI, that offers a variety of spatial functions), as an WPS service. *ArcGIS* server support only the WPS 1.0.0 version of the standard ESRI [2019a]. For someone to use the *ArcGIS* products needs to purchase a licence.

Figure 4.1: ESRI logo

4.1.2 Degree

Degree is an open source software for spatial data infrastructures and the geospatial web, that is written in Java. The software is build on the standards of the OGC and

the ISO, and among the OWS that implements, it implements also the *WPS standard 1.0.0* (offers *GetCapabilities*, *DescribeProcess* and *Execute* operations). Degree supports KVP, XML, and SOAP requests, *asynchronous executions* (with polling of process status) and provides Application Programming Interface (API) for implementing processes in Java Degree [2019].

Figure 4.2: degree project logo

4.1.3 52 ° North

The *52 ° North* is a non-profit organization that provides free and open-source innovative software for the spatial information infrastructures. One of this projects is the implementation of the OGC WPS specification 52 North [2019a]. The implementation is based in *Java* and offers full implementation of the back-end side of *WPS 1.0.0* standard, and a basic client implementation for accessing the WPS including the complete XML encoding 52 North [2019b]. The 52 ° North client side implementation besides the requests against the WPS 1.0.0, supports also requests for the *WPS 2.0.0* (*GetCapabilities*, *DescribeProcess*, *Execute*, *GetStatus*, *GetResult*) 52 North [2019c].

52 ° North supports synchronous and asynchronous invocation with both HTTP-GET and HTTP-POST requests, and is possible to store the execution results 52 North [2019b].

Figure 4.3: 52 ° North logo

4.1.4 GeoServer

As it was already mentioned in the Section 3.3, *GeoServer* can support, besides the other OWS also the WPS as an extension. The main advantage of *GeoServer* WPS implementation over a standalone WPS, is its direct integration with the other *GeoServer* services and the data catalog, making possible to get the input datasets and store the input datasets from and at the *GeoServer* directly GeoServer [2019c].

GeoServer implements the WPS 1.0.0 version of the standard and supports the *GetCapabilities*, *DescribeProcess* and *Execute* operations. Apart of these, *GeoServer* offers as a pseudo-operations the *GetExecutionStatus* operation which generates an *executionsId*. This id can be used by the *Dismiss* operation in order to cancel the execution of the process GeoServer [2019b]. In Figure 4.4 an example of how the status report of the process is given in the *GeoServer*. The progress of the report is given as the number of lines that have been buffered.

4.1.5 ZOO-Project

ZOO-Project is a WPS implementation written in C, Python and JavaScript. It is an open source platform which implements the WPS 1.0.0 and WPS 2.0.0 standards of OGC (*GetCapabilities*, *DescribeProcess*,*Execute*, *GetStatus* and *Dismiss*. *ZOO-Project* is the first and perhaps the only full implementation of the WPS 2.0.0 version of the standard. It supports synchronous and asynchronous execution and provides in

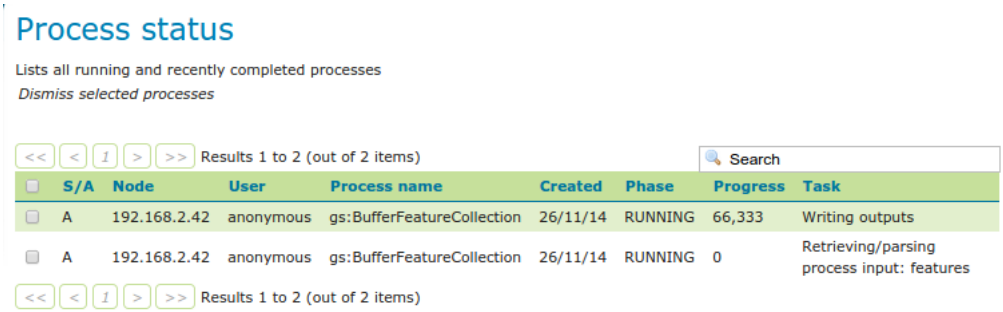

Figure 4.4: GeoServer, status report of the process, Source:GeoServer [2019]

general a developer friendly environment for the creation of WPS. The *ZOO-Project* platform is made up of the following components team [2019]:

- ZOO-Kernel: A WPS server able to manage and chain WPS services
- ZOO-Services: A collection of ready to use WPS built on top of reliable open-source libraries such as *GDAL*, *GRASS GIS* e.t.c.
- ZOO-API: : A server-side JavaScript API for creating, chaining and orchestrating the available WPS.
- ZOO-Client: A client side JavaScript API for interacting with WPS servers and executing standard requests from web applications

4.2 E-GOVERNMENT STANDARD FRAMEWORK OF SOUTH KOREA

Apart from the ZOO-Project the full implementation of the WPS 2.0 standard is still undergoing for almost all the available free or commercial software implementations, and it was difficult to find GIS web projects that publish their geoprocesses under the WPS 2.0 version of the standard. Nonetheless, the e-Government Standard Framework of South Korea that provides a basic environment standards required for the development of web services systems for public projects, has developed a trial system that follows the WPS 2.0 standard. The implementation of the WPS has been made with the ZOO-Project. In this system a WPS that performs satellite image processing has been developed. In Figure 4.5 an example asynchronous execution of the process is given, with the status report Gooseon et al. [2017].

4.3 OPENEARTH INITIATIVE

As it was mentioned during the motivation of this research, the RI2DE tool is a project of the *OpenEarth* initiative of *Deltares* institution. *OpenEarth* (`https://www.openearth.nl/`) started together with *TU Delft*, and motivated by the need of earth science and hydraulic engineering problems for accessibility to high quality data, models and tools. By providing a platform based on open standards and free and open-source technology components, *OpenEarth* succeeds to disseminate and archive high quality datasets, state-of-the-art model systems and well tested tools for practical analysis, breaking that way the artificial boundaries among projects and organizations De Boer [2019].

Aiming to long-term collaboration projects, to re-usability and to developing projects that heritages from others, *OpenEarth* use open standards like the OGC and

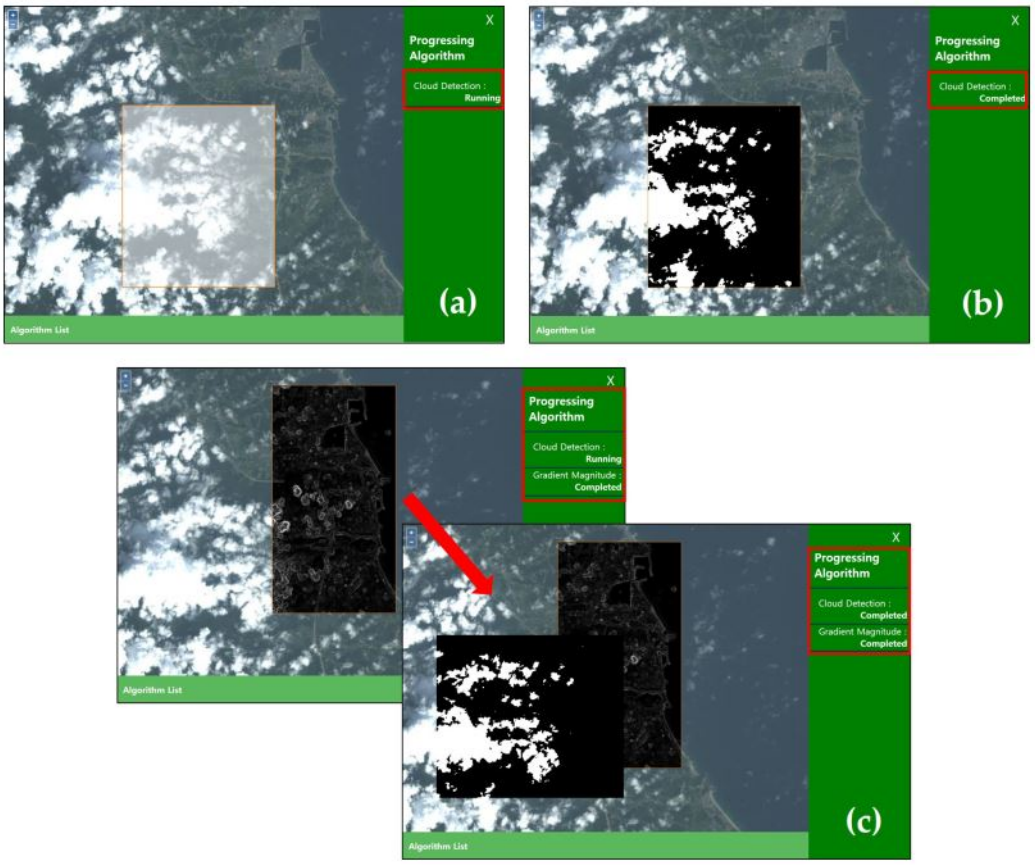

Figure 4.5: Application of the satellite image processing: (a) running process: GetStatus request, (b) completed process and result GetResult request (c) multiprocessing and results

ISO, and promotes the use of free and open source softwares (such as the *PyWPS* implementation of the WPS, the *GeoServer* for publishing raster and vector datasets as services, and the *GeoNetwork* that supports the CSW for creating the Deltares institution different Geoportals).

4.3.1 MI-SAFE Viewer

MI-SAFE tool is a project of the OpenEarth initiative of Deltares, and a product of the Foreshore Assessment using Space Technology (FAST) European Union (EU) FP 7 project. The tool is build ub with several opens srouce components according to OpenEarth philosophy. These components are a GeoServer, a GeoNetwork and a Viewer de Boer and Smits [2017]. The free online tool gives a first indication of the presence and potential flood risk reducing effects of foreshores. The philosophy of the tool is to provide better results in the field sites, by using dataset with high resolution that are extracted from national open data centers. The datasets are already hardcoded in the processes, and when the user selects an area the regional dataset is used Calero [2017]. The philosophy of the MI-SAFE to use regional datasets was the inspiration for the CSW integration in the RI2DE architecture, in order to have a non fixed solution, but a configurable choice of the selection of the dataset according to the user needs.

4.4 RI2DE GIS WEB TOOL

As it was mentioned at Section 1.1 the RI2DE GIS web tool of Deltares institution, which is a project of the OpenEarth initiative is the motivation and the case study for this research. The RI2DE tool performs GIS analysis based on global spatial datasets in order to indicate which areas around road infrastructures are going to be in danger against climate-related geohazards. RI2DE is a tool that targets network infrastructure authorities and consultants that are involved in the designing and maintenance of national or local infrastructure networks. The main idea behind the creation of the tool was to create an open GIS-web platform easily accessible to any stakeholder worldwide that produce geo-hazard analysis maps for detailed susceptibility assessment in regions around the infrastructure network. When this research started, the *Deltares* institution had already developed a first prototype of the RI2DE and produced it online under the *OpenEarth* initiative. The tool is available at (`https://ri2de.openearth.eu`). Since this research project is over, at the URL *link* of the tool it is now available the new version of it.

As the design and development of a new improved version of the *RI2DE* tool is going to be the main goal of this research, it is necessary to describe in detail the technology, the methodology, and the architecture components behind the tool, in order to address the gaps and the problems of the first version of the tool, that made this research a necessity. Section 4.4.1 will describe the **m!** (m!)etroadapt hodology, that the vulnerability analysis steps of the RI2DE were based. Section 4.4.2 will outline the OGC Web Services Architecture of the RI2DE tool and the components that comprise it. Section 4.4.4 describes the datasets and WCS that the tool uses, while Section 4.4.5 illustrates with schemas and detailed descriptions the WPS that comprise the GIS analysis part of the tool. Section 4.4.6 explains how the configuration files of the tool work. Section 4.4.7 present the GUI and the workflow of the user actions on the tool.

4.4.1 Roadapt Vulnerability Assessment methodology

Within the *Deltares* team Climate Resilient Infrastructures, a new vulnerability assessment method for climate-related geo-hazards has been developed, as a part of the *ROADAPT (Roads for today, adapted for tomorrow) project*. The basis of this methodology is that the geo-hazards presents a great variety of related vulnerability factors, and by analyzing and mapping these factors in a GIS environment, we can have as output spatially distributed geo-hazard maps with vulnerability index scores along a section of an infrastructure network, for a specific area Falemo et al. [2015].

The vulnerability factors can be separated to contextual site factors and infrastructure intrinsic factors. The contextual site factors create vulnerability according to the site conditions, vegetation, topography, geology, hydrography etc., while the infrastructure intrinsic factors describe the infrastructure related vulnerability, e.g pavement, road embankment, drainage systems. According to the geo-hazard, some of the factors are a prerequisite for the vulnerability, while others are potential aggravating factors that can increase the vulnerability Falemo et al. [2015].

The ROADAPT VA methodology propose three steps for the creation of the final vulnerability map. But the starting point of this methodology is the selection of the type of the geo-hazard that the vulnerability analysis will be made on and the selection of the study area. An outline of the three methodology steps of ROADAPT VA is presented below:

1. **STEP 1: Defining vulnerability factors**

   The aim of this step is to define what vulnerability factors are going to be used, for the vulnerability analysis assessment of the selected geohazard, from a proposed table of both contextual site and infrastructure intrinsic factors for

a variety of geo-hazards, as it was defined from the *ROADAPT project* team. The definition of the factors should not be restricted to this list, other factors can be selected or the proposed ones can be modified. In an example derived from the *ROADAPT PART C* documentation, the threat Erosion of road bases will be assessed based on the Geology, slope angle, Observed erosion, Existing erosion protection barriers, Land cover, culvert, inspection interval and hydrography (see Figure 4.6).

2. **STEP 2: Data collection**

   The aim of this step is to discover and collect the necessary datasets for the GIS analysis (*STEP 3*). The number of the datasests (layers) will be the same as the number of the vulnerability factors. For instance for the slope angle vulnerability factor, an elevation map should be collected. The format of the datasets is not restricted, so for the elevation someone can use a DEM or contours. The quality of the selected datasets depends on many factors, e.g on the availability, the demands on resolution, the scale, the kind of the threat, the size of the selected area. For transnational studies it is recommended to use either the transnational GIS datasets proposed from the *ROADAPT project* or datasets provided by *INSPIRE directive*. For local or national studies, datasets with higher resolution and with more attributes data can be found. The source of the dataset is based also on the kind of the vulnerability factor. Datasets for contextual site factors can be provided from international or governmental organizations, regional authorities or private companies, while datasets for infrastructure intrinsic factors are most often produced and owned by the infrastructure network owners and its associated organizations.

3. **STEP 3: GIS analysis**

   The core of the ROADAPT VA is the third step where the datasets are analyzed in order to display their respectively contribution to the overall vulnerability. Prerequisite for the analysis is that all datasets should be in the same Coordinate Reference System (CRS) and to be in a raster format. If the datasets don't have the same grid size the highest resolution is kept and the calculation happen based on that. The GIS analysis is happening in two parts. In the first part the datasets of each vulnerability factor are processed and reclassified in three different vulnerability classes:

   - +2 for considerably increased vulnerability
   - +1 increased vulnerability
   - 0 does not increase vulnerability

   For instance in order to produce the raster map with the vulnerability scores of the slope of angle factor, the elevation datasets should be processed and transformed to slope, and later reclassified into the three vulnerability scores according to the desired classes boundaries (e.g less that 1:3 is 0, between 1:15 and 1:3 is +1 and more than 1:1.5 is +2). An example vulnerability scores table for the Erosion of road bases example that was given in the Step 1 is presented in the Figure 4.6

   The second part of the GIS analysis aims to produce the final vulnerability map of the selected geo-hazard. An overlay process of the classified factor layers is proposed, where each raster cell is formulated according to the Equation 4.1. It is possible to weigh the calculation by adding weights to each vulnerability factor layer. The weights should be from 0 to 1.

$$VI = \frac{\sum_{i=1}^{n} VSn}{\sum_{i=1}^{n} VSmax_n} * 100 \tag{4.1}$$

Vulnerability factor	Vulnerability score		
	0	+1	+2
Geology (soil type in natural soil)	Material with low sensitivity to erosion (sedimentary rock, till, clay)	Somewhat erosive material (gravel, coarse sand, silty till, clayey silt, silty clay, peat)	Highly erosive material (fine and medium sand, silt, flood-plain deposits)
Topography/slope angle	less than 1:3	1:1.5 - 1:3	more than 1:1.5
Observed erosion	No	-	Yes
Existing erosion protection	Yes	-	No
Land cover / vegetation	Forest, built-up areas, paved surface, dense vegetation	Arable land, scarce vegetation, solitary trees	disused arable land, other open land, very scarce vegetation, bare soil
Culvert/drum	No culvert or drum crossing road	-	culvert or drum crossing road within 20m from point of evaluation
Inspection interval	Road is inspected more than once per 1 years	Road is inspected every 2-5 years	Road is inspected less than once per 5 years
Hydrography	Distance to watercourse is more than 300m	Distance to watercourse is 100 - 300m	Distance to watercourse is less than 100m

Figure 4.6: Vulnerability scores for threat Erosion

Where:

VI = vulnerability index ($0 \leq VI \leq 100$)

VSn = vulnerability score for layer n

VSmaxn = maximal possible vulnerability score for layer n

n = number of vulnerability factor layers

Final the output of all the steps is a colored vulnerability map with values varying from 0 to 100. An example final vulnerability map of the erosion of road bases threat that was presented in the Step 1, is depicted in the Figure 4.7.

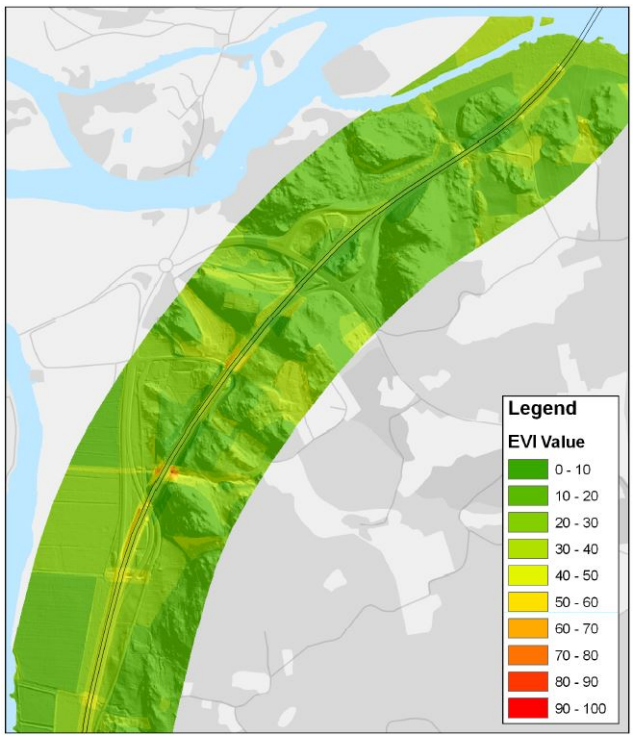

Figure 4.7: Final vulnerability map for the Erosion threat

Threats	Susceptibility Factor
Erosion of culverts	Slope
	Culverts presence
	Distance to water
	Soil type
Landslides	Slope
	Land use
	Distance to water
	Soil type

Table 4.1: RI2DE Susceptibility Factors for each threat

4.4.2 OGC Web Services Architecture of the RI2DE tool

The starting point for the development of the RI2DE tool has been to decide for which type of geohazards and for what areas susceptibility analysis will be offered. The first idea was to create a prototype of the tool, that will offer susceptibility analysis in the *Albania* country with the use of regional datasets, but when the tool started to be developed, the scientific and developers team of the tool decided to make it more generic and work in a global scale. Keeping in line with the steps of the ROADAPT VA methodology the first version of the RI2DE tool offers GIS processes for susceptibility analysis based on transnational datasets for the geohazards, *Erosion of culverts* and *Landslides*. The datasets are open and free disseminated from the organizations as WCS, while the GIS processes (classification of the datasets and susceptibility analysis) are open source and published as WPS from *Deltares* Institution. In Table 4.1 are presented the susceptibility factors for each type of geohazard.

In the architecture schema that is depicted in the Figure 4.8 we can see that the RI2DE tool is composed from many levels of components. In the base level we can see the distributed repositories of the organizations where the raster datasets are stored and the database with the Open Street Map (OSM) features. In the second level are placed the WCS of the rasters and above of them we can see a temporary level of WMS, for the visualization of the result of the processes. The WPS levels represents the top levels of the architecture, as the tool sends request mainly to them. It should not be neglected the most important piece of the RI2DE tool, the configuration files level, where all the information regarding the credential of the database, and the necessary information regarding the location of the distributed WCS is stored. The concept behind the configuration files, is to provide a tool easily configurable and adjustable to the needs of the user.

The depicted architecture of the RI2DE tool can become also a good conceptual flowchart of the functionalities of the tool. More specifically we can see that the client (web-browser) firstly sends request to the "WPS initial" and the "WPS for the selection of Infrastructures", which read the configuration file and connect to the road database respectively, in order to send to the client the information that needs to call the rest WPS. In a second level of actions, according to the selected threat, the client calls simultaneously all the classification WPS of the threat, and the classified maps are stored temporary as WMS at the RI2DE *GeoServer*, where the client (web-browser) access them for visualization purposes. The last step of the RI2DE functionality is to call the WPS of the selected hazard, with inputs the temporary WMS of the classified WCS, and publishes the resulted susceptibility map as WMS to the RI2DE *GeoServer*. Then the client connect to the *GeoServer* in order to get the WMS of the susceptibility map for visualization.

Detailed information regarding the datasets and coverage services, the geoprocessing services, and the configuration files is offered respectively to the Section 4.4.4, Section 4.4.5, Section 4.4.6.

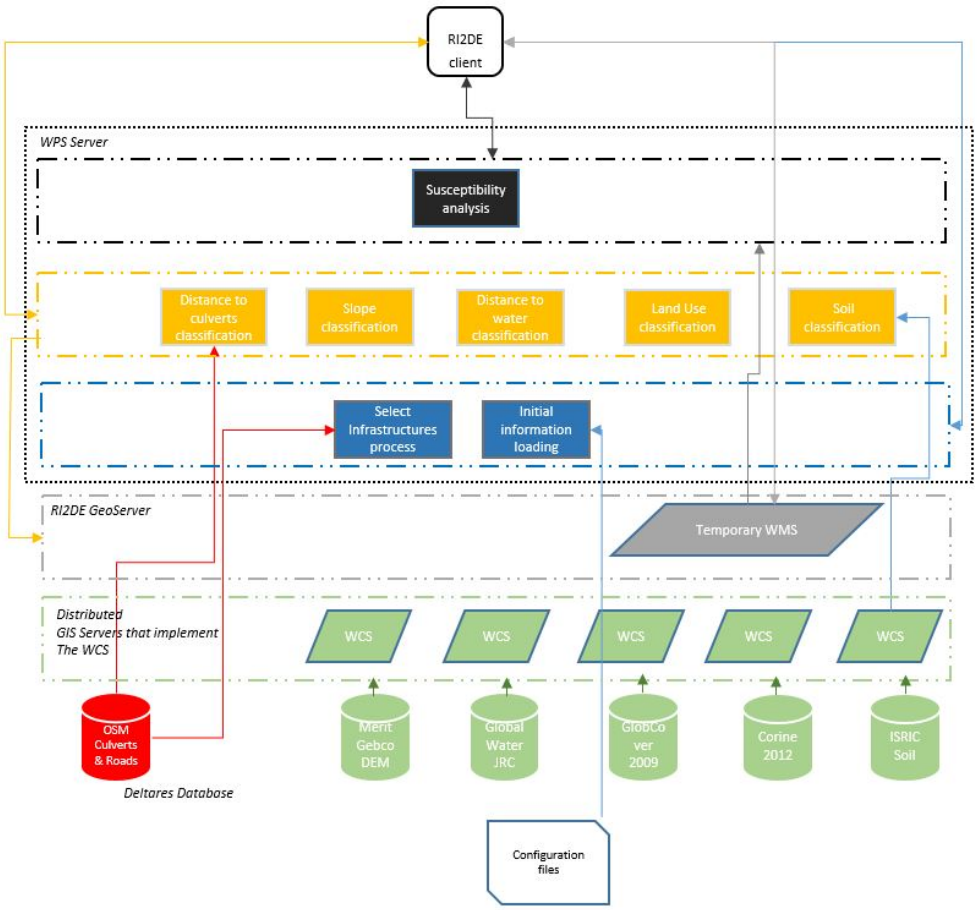

Figure 4.8: OGC Web Services Architecture of RI2DE tool

4.4.3 Technology components of the tool

In Chapter 3 the different technology components of the tool are being analyzed in detail. At this section it will be briefly referred what technology tool has been used for the implementation of the the RI2DE architecture.

All the WPS of the tool are stored in an SubVersion Repository (SVN) repository and the codes for the client side (web browser) are uploaded on *GitHub*. The datasets are mainly provided by other global organizations, but those that comes from the *Deltares institution* are stored also on the SVN repository of the project that were created for. For the development of the WPS, the *PyWPS 4.0.0* has been used. For the publishing of the temporary results of the different GIS analysis processes, as WMS, the *GeoServer* has been used. The *GeoServer* runs on an Apache server. The OSM datasets are stored in a *PostGIS* database. Finally the client side (web browser) has been developed with the *Vue.js/Vuex* JavaScript framework, and runs on a *nodejs* server. In the Figure 4.9 are depicted the different technology components of the ri2de tool.

4.4.4 Input datasets & services

At this section the different services and datasets that were used as inputs to the GIS susceptibility analysis is going to be presented. All the datasets and services that were selected are open and disseminated from international organization, easily accessible to anyone. Below a list with the datasets and services is presented with information regarding, their resolution, origin and location to find them.

- *BASE MAP*

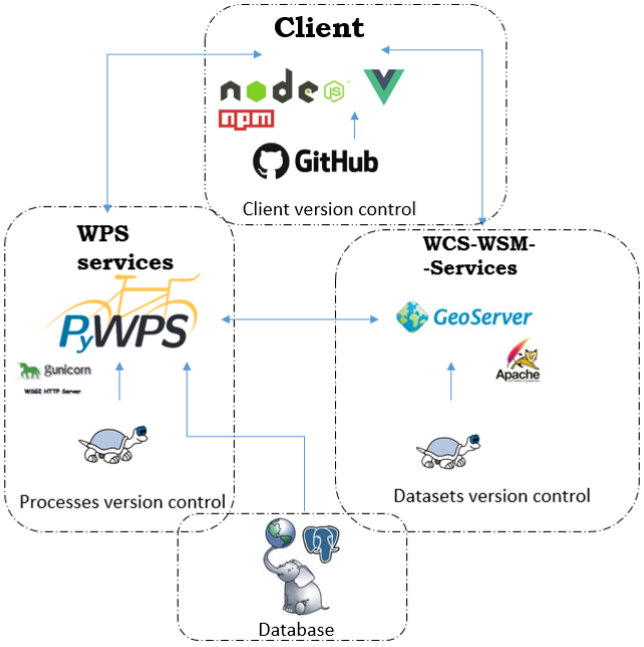

Figure 4.9: Technology components of RI2DE tool

The base map of the RI2DE client (web browser) derives from the *Mapbox*. *Mapbox* is a provider of custom online maps and libraries for websites and applications. The data that the *Mapbox* uses are mainly open source, such as OSM and *NASA* Mapbox [2019].

- *INFRASTRUCTURE NETWORK*

  The basic dataset for the RI2DE tool is the infrastructure network. All the GIS susceptibility analysis that the tool offers starts from the selection of the infrastructures of the desired area. For that reason a detailed and frequently updated infrastructure network with global coverage was needed. OSM, a voluntary open content map initiative, offers feature datasets, open and publicly available to download as points, polylines and polygons Wikipedia [2019c]. Within the *Deltares*, features concerning the different infrastructures (roads, railways) have been downloaded queried and stored in the *Deltares PostGIS* database as shapefiles. The road network is used for the GIS analysis directly from the database.

- *CULVERTS DATASETS*

  In order to classify the area according to the distance to culverts, information regarding the culverts in the area is needed. The culverts dataset that the RI2DE tool use as input in the culverts classification process, are the global culvert lines of OSM. With similar procedure as the infrastructure network, the culvert lines were extracted and stored in the *Deltares PostGIS* database as shapefiles. The dataset is used directly from the database.

- *ELEVATION DATASETS*

  For the slope susceptibility factor reclassify process, raster elevation information is needed. For the slope classification the *MeritGebco* DEM was used, a product of the combination of two different datasets *MERIT* DEM and *GEBCO* bathymetry dataset. The Multi-Error Removed Improved Terrain (MERIT) DEM was created and released in 2017 from the University of Tokyo, by removing multiple error components from the existing DEM (Shuttle Radar Topography Mission (SRTM) and *AW3D*). It has global coverage and the resolution is 90

m YAMAZAKI [2018]. The *GEBCO* global gridded bathymetric datasets are global terrain models for ocean and land, at 15 arc-second intervals Gebco [2019]. For the needs of *Deltares' Fast OpenEarth* project, these datasets were processed and combined, and the *MeritGebco* DEM was created, with global elevation coverage in both ocean and land, and resolution in 90 m. The *MeritGebco* is published as WCS at *Fast OpenEarth GeoServer* of *Deltares*.

- *SOIL DATASETS*

  For the soil susceptibility factor reclassify process, raster soil information is needed. For that reason soil datasets were used with global coverage and resolution at 250 m from International Soil Reference and Information Centre (ISRIC) organization. ISRIC is a world organization with a vision to provide reliable and free soil information. In ISRIC data portal, which is compatible to the CSW protocol, someone can find soil datasets and services. For the purpose of the acri2de project the *SoilGrids* datasets were used. *SoilGrids* provides predictions for percentage of type of soils (*Silt, Sand, Clay*) in seven depth layers.The datasets are available for download at *ISRIC* repository system, but they are also available as WCS Hengl et al. [2017].

- *LAND USE DATASETS*

  For the land use susceptibility factor reclassify process, two different land use WCS were used an inputs, depending on the location. If the desired area for susceptibility analysis is in Europe the *Corine 2012* is used, while if it is somewhere in the rest of the world the *GlobCover 2009* is used. Both of the datasets are products of European Space Agency (ESA) and are published for free use. More specifically the *GlobCover* products have been processed by the ESA together with the *Université Catholique de Louvain*, and provides land cover map with global coverage, generated by an automate process chain from the 300 m *MERIS* time series ESA [2019].

- *WATER DATASETS*

  For the water classification process, the input dataset is a surface water raster with 25 m resolution, from the *JRC* water portal. The *JRC* water portal provides access (for view and download) to water data.

4.4.5 Web Processin Services

The functionality of the RI2DE tool is based on a number of WPS that perform:

- Loading of the hazards' information
- Selection of the roads
- Classification of the datasets
- Susceptibility analysis

At this section a detailed overview of the different processes will be carried out, by analyzing the way they work, the inputs they accept, and the outputs they produce. All the processes are published as WPS. This analysis will give a good insight on the flexibility and performance of the services. For the GIS analysis that is performed in the processes is used the on thFor the presentation of every WPS a small description will be given on the way they perform, followed by a schematic view for better understanding.

Susceptibility Factor	Susceptibility score			
	1	2	3	
Land use("GlobCover 2009")	20, 30, 50, 60, 90, 100, 120, 170, 220	14, 40, 60, 90, 100, 130, 140, 160, 180	11, 150, 190, 200, 210, 230	Non configurable from the user
Land use ("Corine 2009")	23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33	15,16,17,18,19,20,21,22	1,2,3,4,5,6,7,8,9,10,11, 12,13,14,34,35,36,37, 38,39,40,41,42,43,44, 48,49,50	Non configurable from the user
Slope (Default values)	<5 degrees	<10 degrees	<90 degrees	Configurable from the user (RI2DE client)
Soil (ISRIC)	sand	silt	clay	Non configurable from the user
Distance to water (Default values)	<50 meters	<100 meters	<1000 meters	Configurable from the user (RI2DE client)
Culverts presence (Default values)	<50 meters	<100 meters	<1000 meters	Configurable from the user (RI2DE client)

Figure 4.10: Susceptibility factors default classes boundaries

Process for the initial loading of the hazards' information

All the information that concerns the hazards and their susceptibility factors, is stored in a JSON configuration file on the same repository with the processes. This WPS is the process that reads this configuration file and sends it back to the client. In the Figure 4.11 it is depicted a schematic view of this process. The configuration file is stored in the same directory as the WPS.

At this point it should be mentioned that for the codes of the different processes the tool offers were created with the python packages:

- The open source *GDAL/OGR* python package for the different raster and vector GIS analysis.
- The *OWSLib* in order to implement the different OWS operations that were needed in the processes (e.g. GetCoverage in order to get the raster datasets from the WCS.
- The *geojson* package in order to manipulate the *GeoJSON* (The polygon and the lines are sent between the client and the server in the inputs and the outputs of the WPS as *GeoJSON*.
- The *sqlalchemy* package in order to connect to the *PostGIS* database and query for the roads and the culverts

Process for the selection of the infrastructures

The WPS for the selection of the road infrastructures, is a process that accepts as input the polygon around the study area in a GeoJSON format, and according to this, it connects to the *PostGIS* database that has the OSM roads and extract the buffered roads and the lines of the roads that falls in this polygon. The size of the buffer is configured in the configuration file. The buffered roads are stored in the temporary directory, while the road lines together with a random roads id that is created in the process, are registered to the response. In the Figure 4.12 a schematic view of the process is depicted.

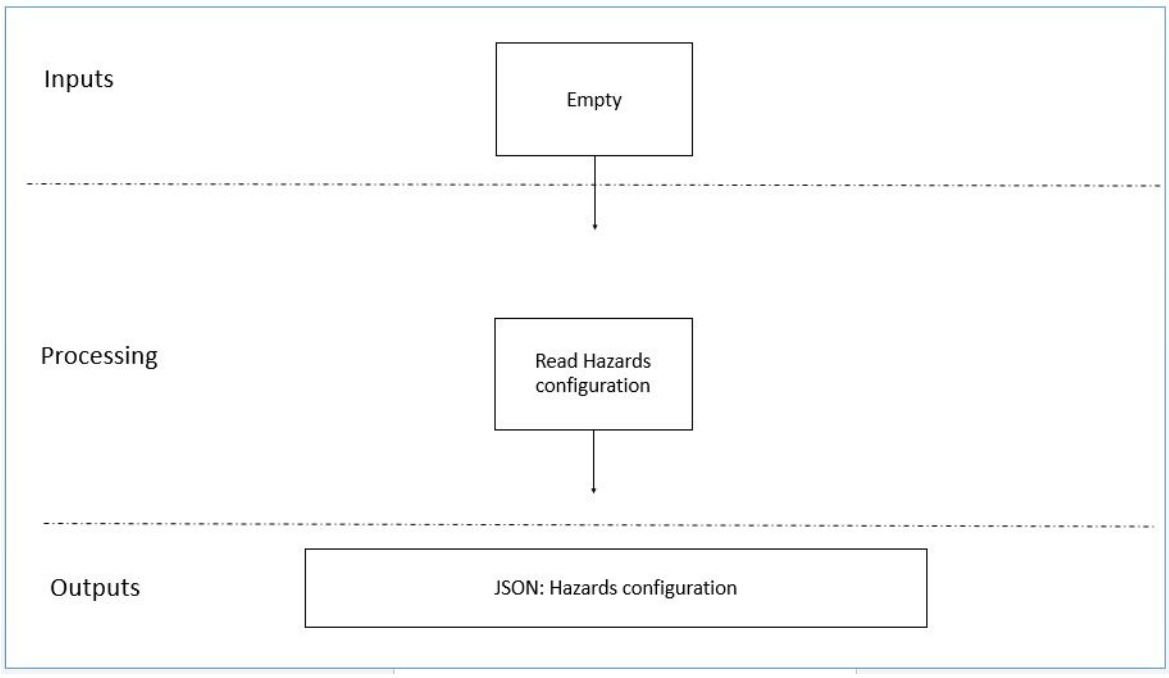

Figure 4.11: Schematic view of WPS: Initial loading of hazards' information

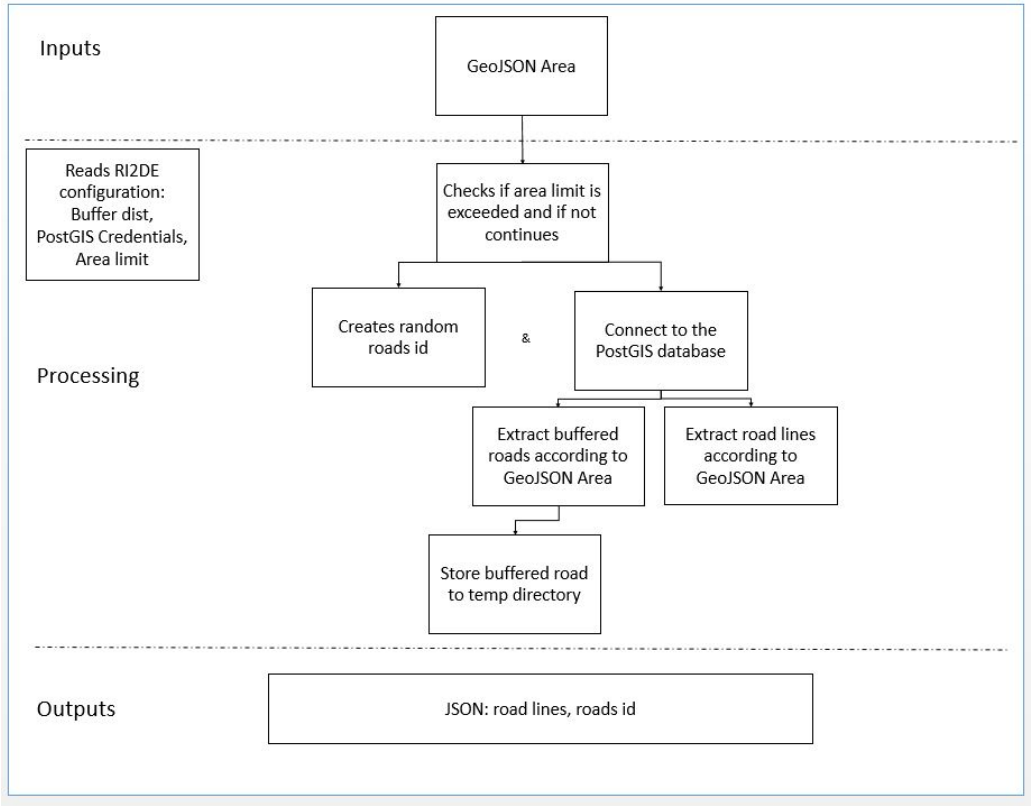

Figure 4.12: Schematic view of WPS: Selection of the roads

Processes for the classification of the datasets

Every classification WPS has been developed on a base of a custom classification WPS. The idea was to have the same sequential steps for all the classification processes. But in some susceptibility factors the classification process is highly dependable on the dataset. For that reason every WPS will be analysed separately, starting from describing the WPS custom steps, and then examining each WPS on its' flexibility.

CUSTOM CLASSIFICATION PROCESS: The WPS custom is not a real process but just a skeleton of the desired steps for the classification of the datasets. An ideal flexible WPS accepts as input the *identifier of the road*, and a *layer setup* that provides: *the boundaries of the classes*, *the layer name* of the WCS on the *GeoServer*, and the *OWS URL* of the server. The first step of the custom process is to read these inputs. Then with the *roads identifier* gets from the the temporary repository the *GeoJSON* of the roads, and with it creates a bounding box. Using the *layer name* and the *OWS URL*, in the next step it connects to the *GIS Server* and implements the *GetCoverage* operation cutting only the area of the coverage service that fell in the bounding box. With the downloaded coverage (raster), applies the reclassification function according to the input classes boundaries. In the last step publish the reclassified coverage as WMS to the temporary repository of the *GeoServer*. In the output are registered in a JSON file the style of the *GeoSerer*, the *layer name* and the *OWS URL* of the temporary WMS. In the Figure 4.13 is depicted a schematic view of the custom WPS. In this schema the different steps of the whole classification procedure have numbers in order to help the reader to understand from which point and after the susceptibility factor WPS of the tool decline from the custom steps. Every susceptibility factor WPS have the same JSON output format and accepts the same format of inputs (*roads id*, and *layer setup*), but depending on the susceptibility factor flexibility, in the *layer setup* either the *classes*, or the *layer name* and the *OWS URL* are empty, as they are not configurable. In Figure 4.10 the default classes boundaries of each classification process are presented. The classes that are configurable have default values in the configuration file of the tool, that can be altered from the web browser later, while the one that have fixed boundaries, have them hardcoded in the process.

DISTANCE TO WATER CLASSIFICATION PROCESS: The WPS that classifies the distance to water susceptibility factor follows the same procedure steps as the custom WPS apart from the part of the classification function. After the 3rd step of the WPS custom, where theWPS connects to the *GeoServer* with the *layer name* and the *OWS URL*, and gets the part of the coverage that fell in the bounding box, the WPS water reclassify the downloaded coverage, with a dilation image technique. This classify function demands the resolution of the dataset, and the resolution is hard coded to 25m, equal to the resolution of the *Global Water* Joint Research Centre (JRC) coverage service (see water datasets at Section 4.4.4). In order to consider the WPS distance to water as a flexible procedure, an extra function should be added that reads the resolution of the downloaded coverage and provides it to the classification function. In the schematic view Figure 4.14 it is depicted the classification function for the distance to water.

SOIL CLASSIFICATION PROCESS: The WPS for the soil susceptibility factor is highly dependable on the input soil dataset, as the classification is applied on the different soil categories that the dataset offers. The soil classification process was developed based on the global coverage soil dataset of ISRIC organization. ISRIC provides predictions for percentage of type of soil (Silt, Sand, Clay) in seven level depths. In the classification process the 7 depths layers of the three datasets (Silt, Sand, Clay), meaning 21 coverages, are downloaded based on the bounding box. Then the reclassification is applied according to the classes, as they are presented on the Figure 4.10, on the overlay of the three soil coverages that has the mean soil values of the seven

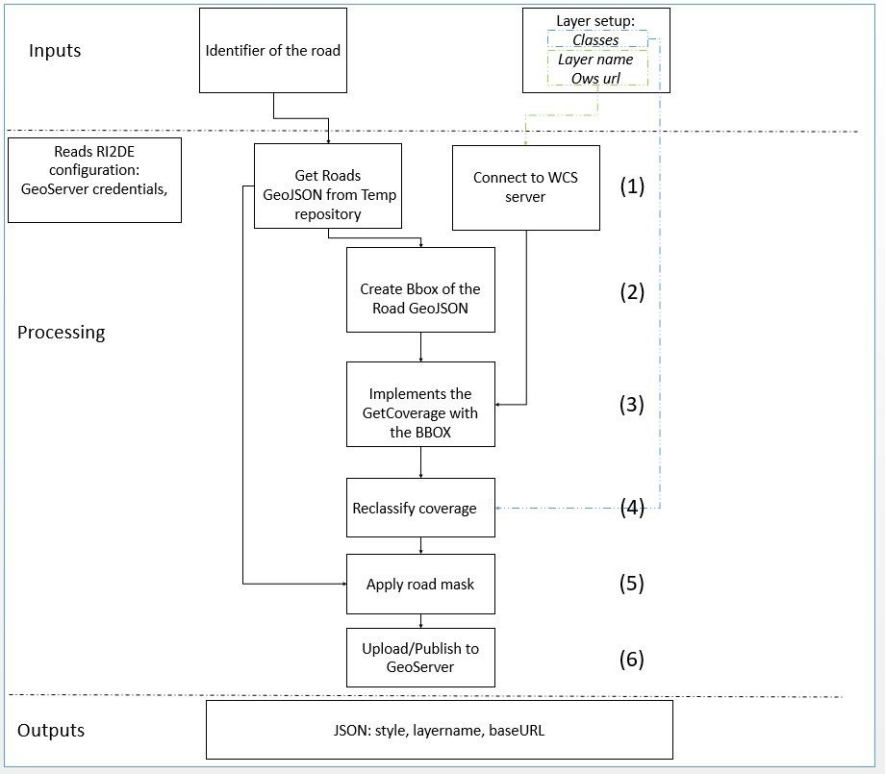

Figure 4.13: Schematic view of WPS: Custom process for classification of datasets

depths. The schematic view of the soil classification process, is depicted on the Figure 4.15. On that image we can see that theWPS has empty values on the *layer name*, *OWS URL* and *classes*, as they are hard coded in the process, and after the creation of the bounding box the procedure declines from the custom process.

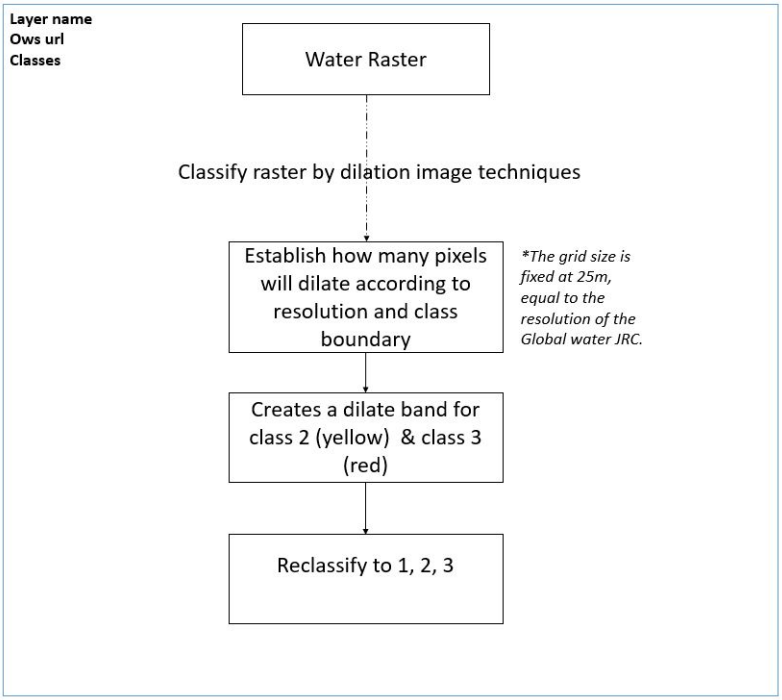

Figure 4.14: Schematic view of WPS: classification process of distance to water

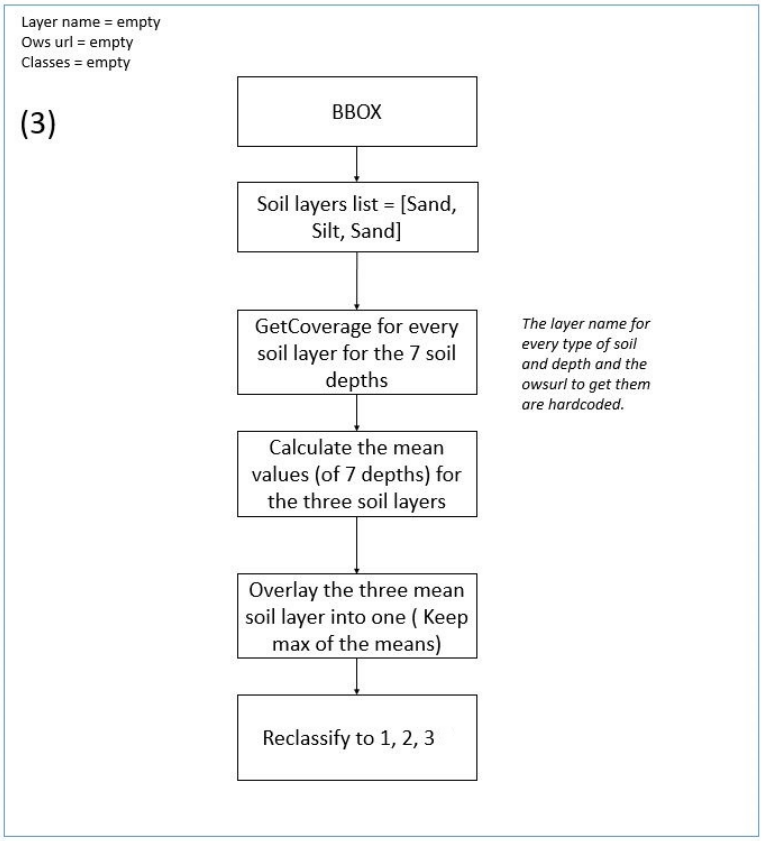

Figure 4.15: Schematic view of WPS: classification process of soil

LAND USE CLASSIFICATION PROCESS: Similar to the soil classification, the land use classification declines from the custom process from the step 3 and after, and the classification function is developed based on the land used datasets *Corine 2012* and *GlobCover 2009*. In the Figure 4.10 are presented the land used categories that were used for the classification, and in the Figure 4.16 it is depicted a schematic view of the land use classification procedure after the creation of the bounding box.

CULVERTS CLASSIFICATION PROCESS: The classification of the culverts differs a lot from the custom WPS. The dataset that was used for the culverts susceptibility factor, is the acosm culverts lines, which are stored in the *PostGIS* database of *Deltares*. So after the creation of the bounding box, the classification algorithm, connects to the *PostGIS* database, and extract the buffered culvert lines that fell into the bounding box. The buffer size depends on the distance values of the class 2 and 3. Then according to the resolution value (is extracted from the configuration file), the buffered lines are rasterized, and then combined with the masked road in order to create the classified culverts raster. In the Figure 4.18 it is depicted a schematic view of the classification procedure of the culvert lines. The *layer name* and the *OWS URL* are empty in the layer setup input, but the classes are provided. The default classes are presented in the Figure 4.10.

SLOPE CLASSIFICATION PROCESS: The slope classification WPS has exactly the same skeleton as the WPS custom. After the download of the elevation coverage, it transforms it to slope with degrees values, and then reclassifies it according to the given classes boundaries. The default classed boundaries are presented in the Figure 4.10, while in the Figure 4.17 it is depicted a schematic view of the classification process. The slope classification process needs all the values of the layer setup, as it accepts any DEM.

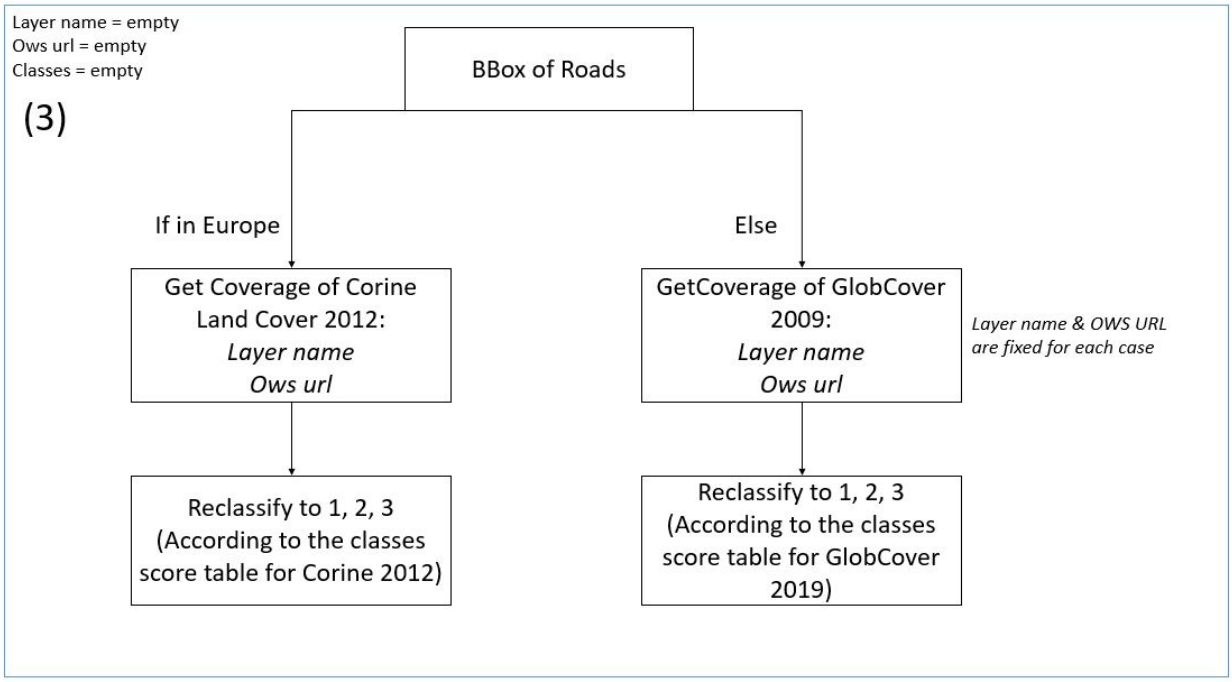

Figure 4.16: Schematic view of WPS: classification process of land use

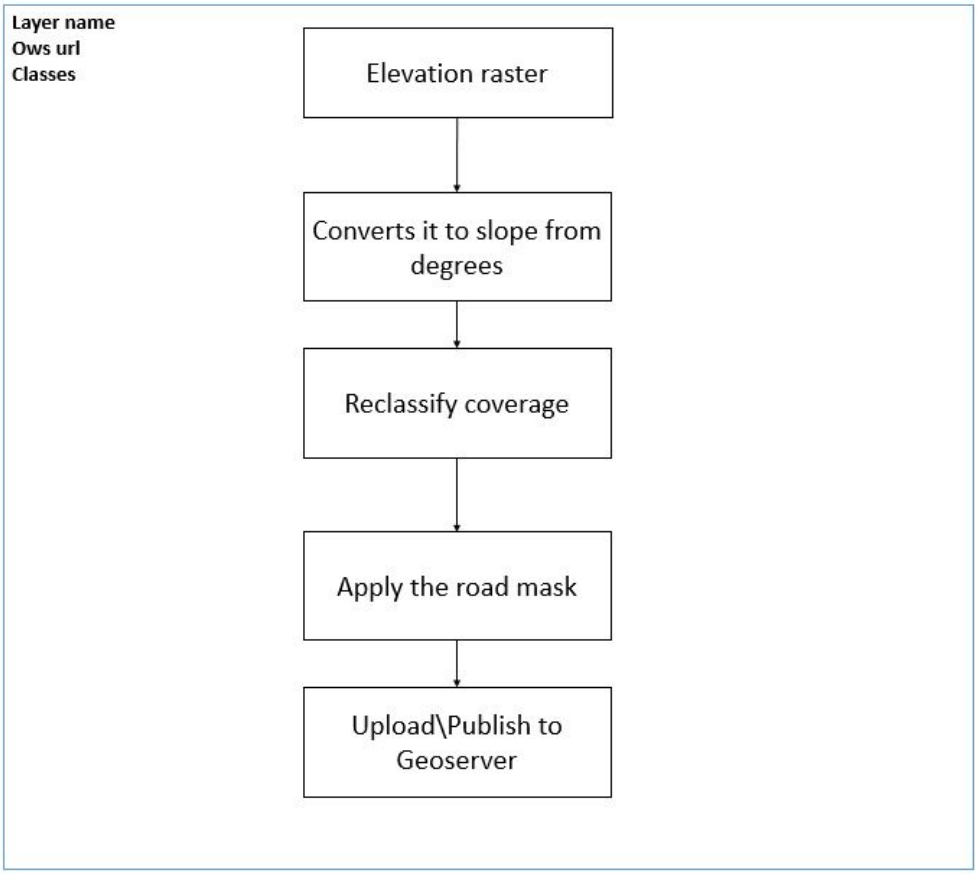

Figure 4.17: Schematic view of WPS: classification process of slope

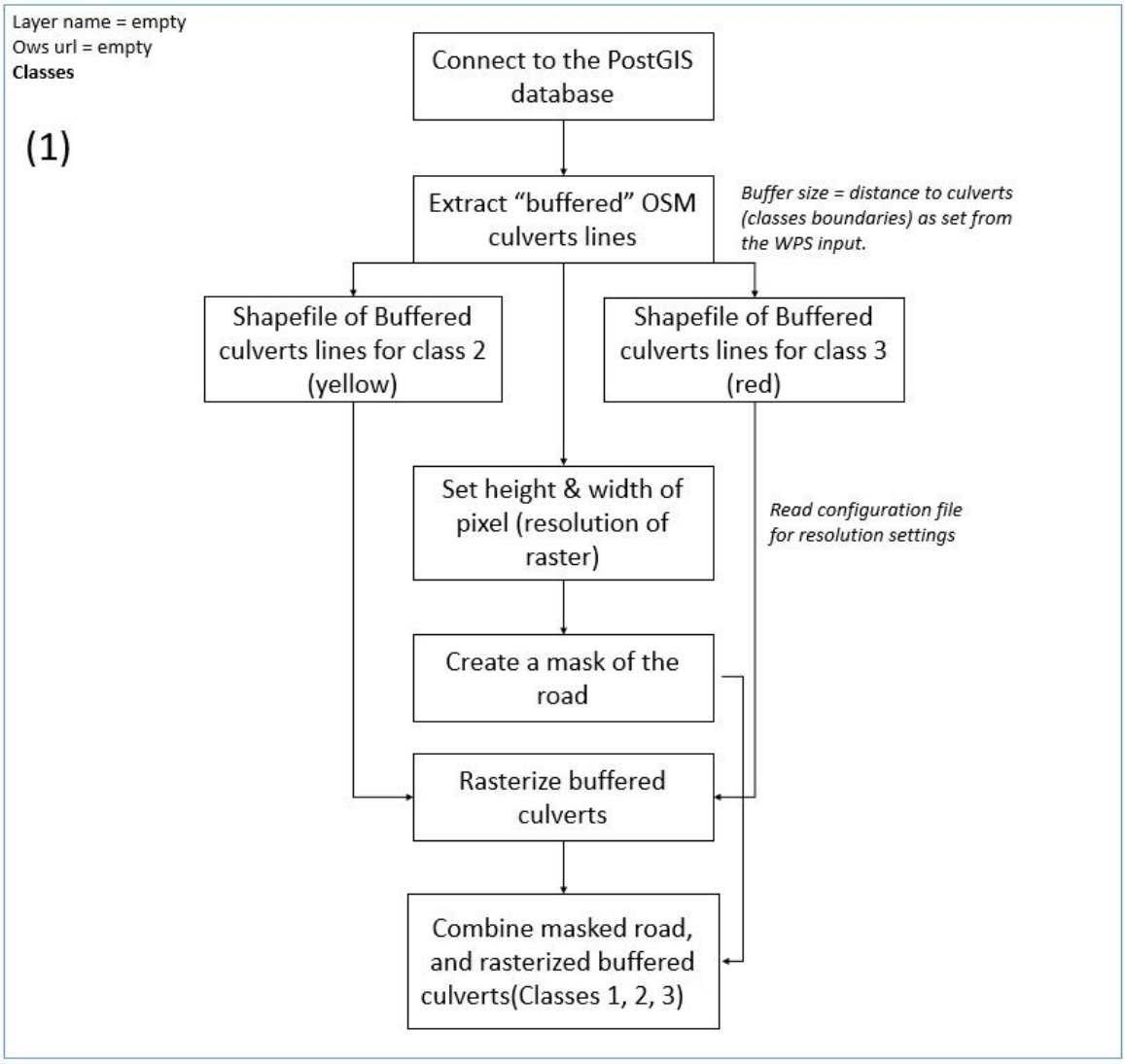

Figure 4.18: Schematic view of WPS: classification process of culverts

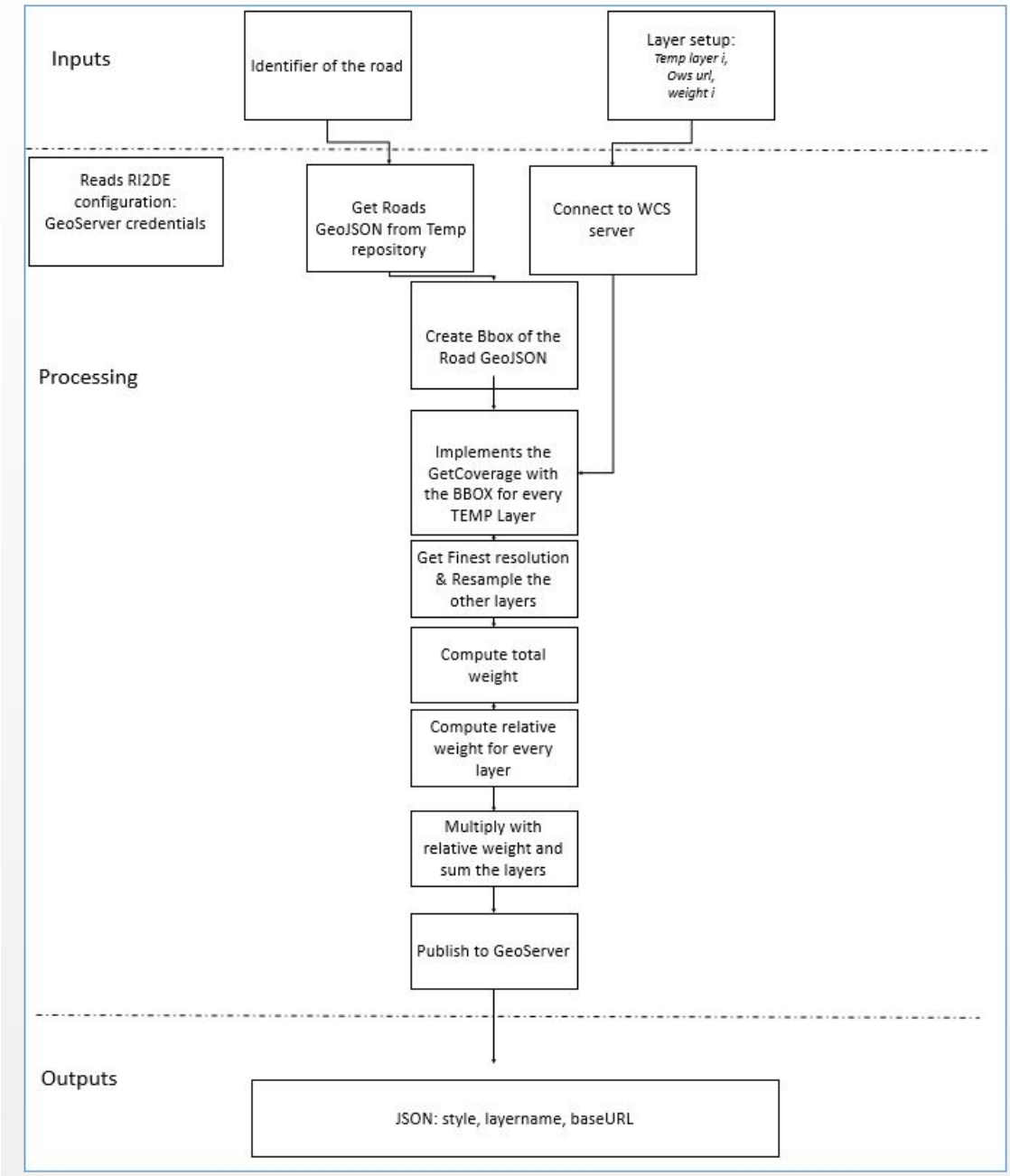

Figure 4.19: Schematic view of WPS that provides the final susceptibility analysis

Susceptibility analysis process

The final susceptibility analysis is a process that is based on the classified datasets for each susceptibility factor, and is the same for every hazard. The WPS that implements it accepts as inputs the road identifier, for the creation of the bounding box, and a layer setup, which contains the *layer name*, *OWS URL*, and weight of every classified dataset. The final susceptibility analysis is developed on the concept that every susceptibility factors has a weight on the calculation of the final susceptibility map, and the values vary from 0 to 1. In the WPS, with the use of the *layer name* and the *OWS URL*, every classified dataset is downloaded. Since the raster datasets don't have the same grid size, the raster with finest resolution is selected and based on its' resolution the rest of the raster are re-sampled in order to create the final raster with grid size equal to the finest resolution.Then the total weight of all the raster is calculated and every raster is multiplied with each relative weight (weight/total weight). At the end all the raster layers are summed in order to get the classified susceptibility layer, and it is published to the *GeoServer* as WMS. The output of the WPS is a JSON file with the style of the GeoServer, the *layer name* and the *OWS URL* of the WMS. In the Figure 4.19 is depicted a schematic view of the susceptibility analysis WPS.

4.4.6 Configuration files

At this section they are described in detail the two configuration files of the RI2DE tool. The concept behind the use of configuration files, is to provide a tool easily configurable and adjustable to any system and to the specific needs of the user and to avoid hard coding in the software. The two configuration files are stored in the same SVN repository as the WPS. The first configuration file is called "ri2de configuration.txt" and contains more generic information concerning the system and some original settings. Detailed description on what information of the RI2DE system the user can configure is presented in the Table 4.2.

Table 4.2: ri2de configuration

Geoserver	
host	Credentials of the Geoserver where the temporal Web Map Coverages are stored/published
ows_url	
wms_url	
rest_url	
pass	
PostGIS	
host	Credentials of the PostGIS database where the OSM infrastructure network and the culverts shapefiles are stored
user	
pass	
db	
port	
Settings	
area_limit	The limit of the size of the area that the user is able to select infrastructures
buffer_dist	This value indicates the size of the buffer of the area that is going to be studied around the infrastructure
res_culverts	The resolution for the creation of the culverts raster in the culverts Web Processing Service
tempdir	The temporal directory to store the GeoJSON of the roads

Table 4.3: RI2DE susceptibility factors setup JSON

E.G. Landslides			
Susceptibility factor i	title	The name of the susceptibility factor is a basic information for the client	compulsory
	WPS function id	The id of the classification WPS of the factor is a basic information for the request of the WPS from the client	compulsory
	classes	The original values of the boundaries of the classes.	optional (In some processes it is empty. Some of the classification processes have the classes boundaries hardcoded)
	layer name	The layer name of the WCS on the GIS Server	optional (In some processes it is empty. Some of the classification processes dont work with WCS).
	OwsUrl	The endpoint url of the GIS Server in order to connect.	optional (In some processes it is empty. Some of the classification processes don't work with Web Coverage Services)

The second configuration file is written is a JSON file in order to be sent easily at the client, and is like a library with all the information concerning the geohazards and their susceptibility factors. More specifically this file contains for every susceptibility factor information concerning the title, the name of its classification WPS, the boundaries of the classes, and the layer name and OWS URL of the server that is published.The knowledge of the client on the threats and the factors is based on this file. An example format of the file is presented on the Table 4.3.

4.4.7 GUI and workflow of the user actions in the RI2DE client

At the last section of the RI2DE tool description it is important to outline the workflow of the user actions with the tool in order to reach to the result of the susceptibility analysis. For the better understanding of the workflow, together with the description a series of snapshots of the GUI of the web browser are depicted.

1. *Selection of the infrastructures*

   When the user open the web page of the tool, there is a start screen with the global map. In this first screen the user can navigate either with the mouse or by the search bar, to the area that has the infrastructures that want to do the hazard analysis, and then zoom in and wait a while for the infrastructures to load. In order to select the infrastructures, the user should click on the polygon tool, create a polygon around them, and then either press enter or double click for saving it. On the left of the screen the user can see in the infrastructure tab the name of the selection that made. The default name is Selection 1, but it can be altered. There is also the option for the user to make extra selections (create more polygon selections on the same way) in order to

assess the vulnerability of multiple areas on the same time. After that the user should click next in order to go to navigate to the next step of the hazard analysis (Figure 4.20).

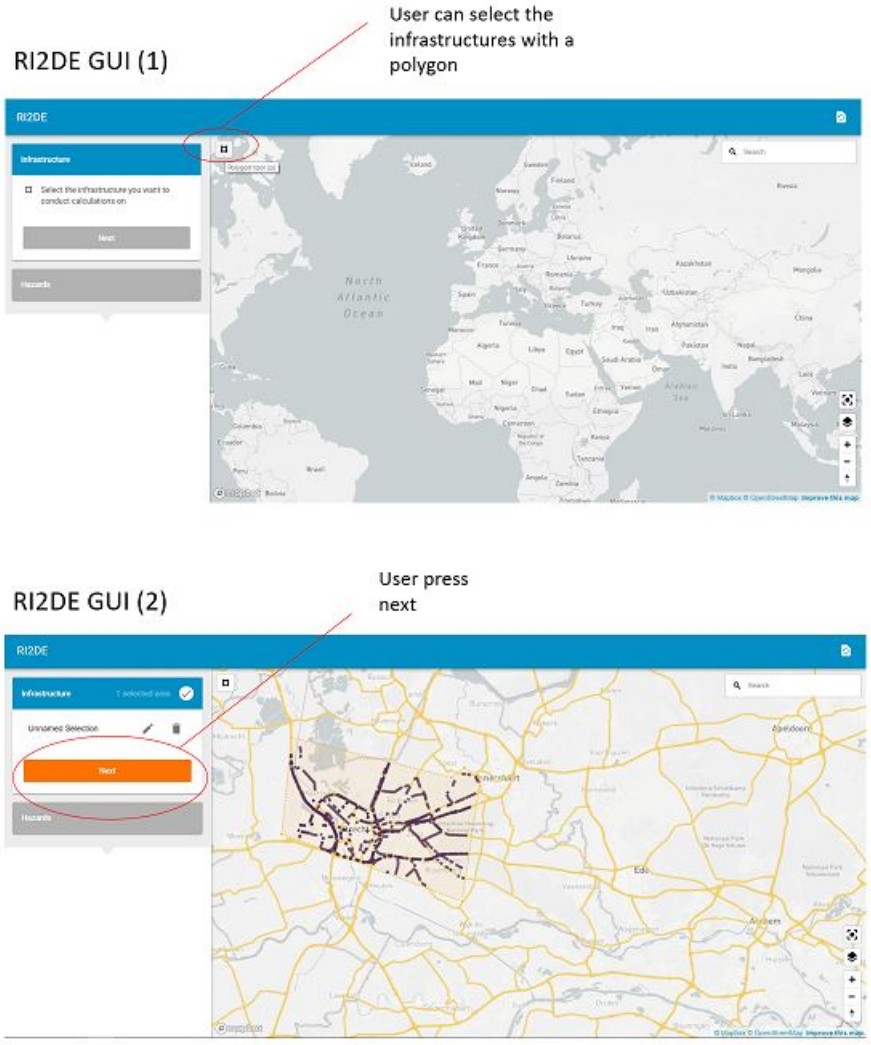

Figure 4.20: GUI of RI2DE tool, part 1

2. *Selection of the type of the hazard*

   In the RI2DE tool web page now at the left it is open the *Hazard* window tab. At this window there is a list with the available hazards for analysis. The tool offers analysis for the Erosion of culverts and landslide geohazards. The user can select the hazard to do susceptibility analysis and then click next in order to navigate to the next tab with the susceptibility factors of the selected hazard IFigure 4.21).

3. *Configure the susceptibility factors*

   In the new tab now on the left of the RI2DE tool there is a list with all the susceptibility factors for the final hazard analysis. The user can either click straight the calculate button in order to produce the final susceptibility map of the selected hazard, with the default settings of the factors, as they are provided from the JSON configuration file that was loaded with the "WPS initial loading of the hazards' information" or alter these setting on a windows that appears by clicking each factor separately. At this window the user can either

adjust the weighting of the factor (the values vary from 0 to 1), or change the boundary of the classes (this option is not available for all the factors) which reclassifies the susceptibility factor map and shows itFigure 4.21.

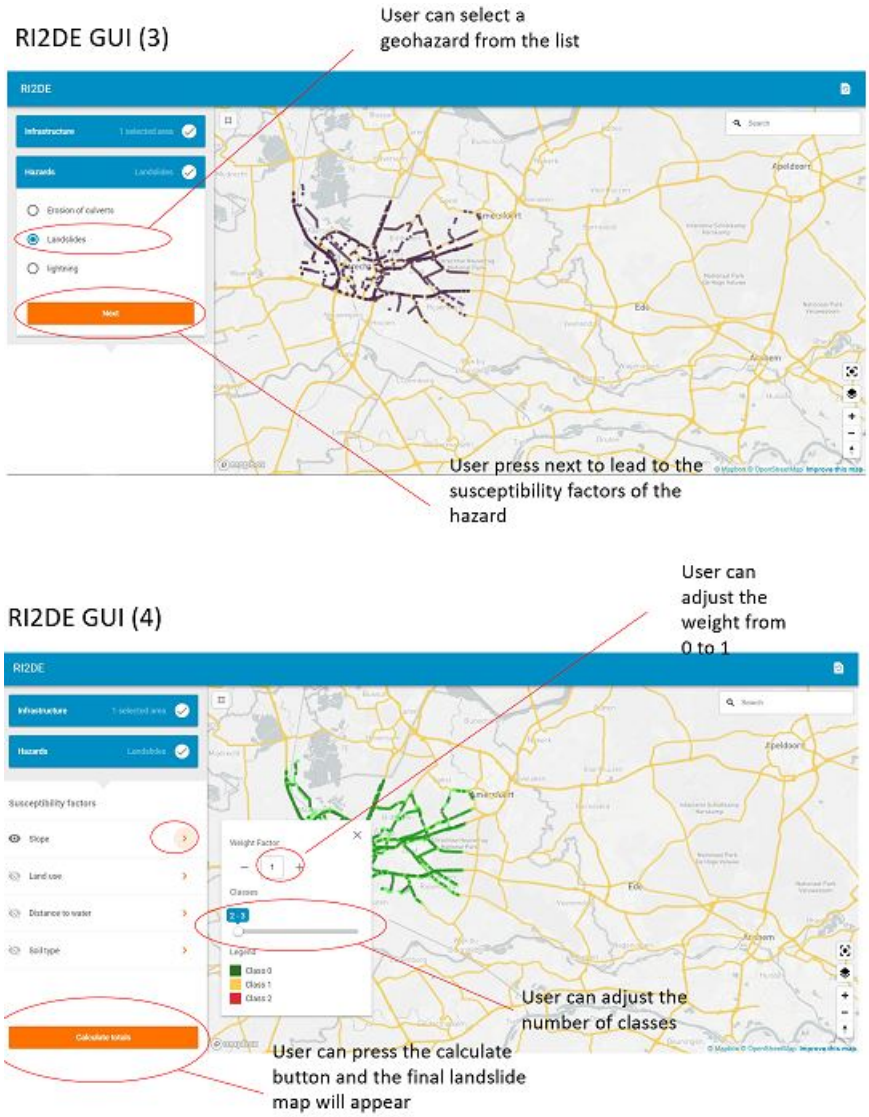

Figure 4.21: GUI of RI2DE tool, part 2

4. *Susceptibility map*

   After the calculation button is clicked the final susceptibility map of the hazard is produced and the user can zoom in and navigate to the area in order to assess the danger around the infrastructure. From that final step the user is able to click either the infrastructure tab in order to create a new selection, or the hazard tab, in order to change either change the hazard to analyze or stay on the same and alter the susceptibility factor settings.

5 REQUIREMENTS

When this research started, a set of new functionalities were decided to be added to the new version of the RI2DE tool, in order to have a susceptibility analysis tool more flexible to the user needs, with a logical workflow to reach to the final result, and that communicates the results or problems-errors better to the user. In order to assess the feasibility to have these new functionalities under an OGC Web Services Architecture, during the analysis framework phase, a thorough research has been carried out:

1. On the WPS standard and its' *PyWPS* implementation that the RI2DE use
2. On the CSW standard and the different metadata formats

The WPS 2.0.2 version of standard offers the *GetStatus*, *GetResult* and the *Dismiss* operation, which under an *asynchronous* execution can provide status reporting and control of the process. Unfortunately the *PyWPS* implementation of these operations is still undergoing.

This indicates from an early phase of the project that it will be difficult and some times also not feasible to implement all the functionalities that were set at the beginning of the project. For that reason at this point of the research it is essential to outline the ideal set of functional requirements that are necessary for the development of the new functionalities, based on the OGC Web Services research. For every functionality it will be indicated if they are going to be developed during the implementation phase of the project or if they are going to a future work.

As it was already mentioned at the research scope of this project, this research is happening simultaneously with the development of some extra functionalities on the RI2DE tool, from the *OpenEarth* developers team. For that reason it is important to make a quick reference on the new functionalities that they are going to add to the tool. Section 5.1 will describe the ideal requirements in order to discover services from the web viewer of the tool, and then use them as inputs to the different processes. Section 5.2 will describe the changes that should be made to the tool in order to support status reporting according to the WPS standard.

5.1 DISCOVER, RETRIEVE, DISPLAY AND USE SERVICES

Public authorities, governments, organizations and institutions in all parts of the world, driven by the concept of open data for everyone, have established SDI (geoportals) in order to freely offer their datasets, tools, and services. One of the most essential parts of an SDI is its catalogue server where they register information about their datasets, processes and services (metadata).

The idea behind the discover, retrieve, display, select and use services functionality, is to connect to different local and national geoportals (e.g INSPIRE) that are developed under the standard rules of the CSW and find and access the services they offer, in order to use them as source inputs in the different classification processes of the RI2DE tool. The idea is to create somehow a "market of datasets" in which the users can see and select according to their needs of quality and resolution.

Follows a list of the functional requirements that are needed in order to develop this new functionality with a brief explanation.

- Discover & Retrieve: Implement the *GetRecords* operation of the CSW communication protocol.

  Implementing the *GetRecords* operation makes feasible to discover resources registered in a catalogue that fulfils the CSW specification, and it will allow us to make a search operation and retrieve the records that we want. The most important part of the *GetRecords* implementation will be the query construction, which should be developed in a way that can be used to search for metadata at any catalogue and to be used by all the susceptibility factors.

- Display, select and use: After the records are retrieved, they need to be displayed to the users in order to select one of them as the new source to the classification process.

  The display part belongs more to the design phase, where the appropriate way to show the records should be decided (e.g. like a market of datasets) , while the select and use part demands to find a way to change the original *state* (see Section 3.4) of the *OWS URL* and the *layer name* on the client (web browser) in order to re-send the request to the classification process with the *OWS URL* and the *layer name* of the new service.

- Reprojection: The changing of the source of the classification process, created an extra requirement.

  So far the services that were used, were projected on the World Geodetic System 1984 (WGS84). Using services from different regional servers that are on the local projection system, arise the need for a reprojection process in the back-end of the tool, in order to have all the services on the same projection system.

5.2 STATUS REPORTING OF THE PROCESSES

In the first version of the RI2DE tool the different processes that it offers (selection of the roads or classification of the datasets), don't take long time to finish, since there is:

- a maximum limit on the size of the selected area, in order to avoid long time running of the process that selects the infrastructures
- the input datasets have global coverage with very low resolution, resulting on fast classification processing

Resetting the limit of the size of the selected area, letting the user to select even whole countries for susceptibility analysis, and offering the option to alter the input datasets to the classification processes, with local high-resolution datasets, may result to long time processing and on the same time, long time of waiting from the user perspective.

For that reason the continuous reporting of the status of the processes to the user, is important for the new version of the RI2DE tool, and the best solution to have status reporting under an OGC Web Services Architecture, is through the new operations (*GetStatus* and *GetResult*) of the WPS 2.0.2 standard.

The functional requirements that need to be fulfilled in order to succeed status reporting through WPS are outlined in the list below, but since the *PyWPS* implementation of the the WPS 2.0.2 is still undergoing, these requirements are not going to be implemented at this project. The implementation ca be a future work, if the *PyWPS* eventually implements them or if the WPS of the tool published with the *ZOO-Project* which offers a full implementation of the standard.

- Alter the Execute request :

  The Job monitoring in the WPS standard is allowed only in the asynchronous execution of the process. In the first version of the RI2DE the execution of all the WPS is happening synchronously, so the first requirement is to alter the execution to asynchronously. This can be succeeded through the Execute request, if we set the *ResponseFormat* parameter to "StatusInfoDocument" and the *Execution* parameter to "asynchronous". On that way when the execute request is going to be invoked from the client, a job id will be created in the WPS server and a *StatusInfoDocument* will be returned containing the job id, the URL to poll the different *StatusInfoDocument* and the final result location.

- Implement the *GetStatus* operation

  The next requirement is to construct a *GeStatus* request that it will be sent from the web browser to the WPS server, from which the *StatusInfoDocument* will be returned every time, with information regarding the status of the job.

- Implement the *GetResult* operation

  The final step is to construct a *GetResult* request and send it from the client, when the processing is done. The *GetResult* request has as a response, the *ResultDocument* which contains the output of the process.

5.3 CONTROL OF THE PROCESSES

The same reasons that arose the need for status reporting of the process in the RI2DE tool, arose also the need for control. So far when a process starts, the user has to wait until it finishes, as there is no way to stop it, apart from refreshing the page and loosing all the parameters that were set. The new version of WPS standard introduced a way to control the process, with the *Dismiss* operation. So the way to stop a process in an OGC Web Services architecture is through this operation. The Dismiss operation will not be implemented, since the *PyWPS* does not support it yet. The implementation can be future work, through the *ZOO-Project* or when the *PyWPS* finish the implementation. The requirements in order to achieve that are the following:

- Implement the *Dismiss* operation

  Create a base XML request for the *Dismiss* operation on the client(web browser), having as variable the job id. When the process starts to run in asynchronous mode and sends the job id, if the user select to stop the process, create the Dismiss request with this job id.

A schematic overview of the desired asynchronous execution is presented in the Figure 5.1

5.4 TRANSLATE TO RISK

As it was already mentioned, the target audience of the RI2DE tool is infrastructure network owners or governmental/ local stakeholders that are deeply involved to the maintenance of roads. For them the main goal for a geo-hazard susceptibility analysis, is to estimate the cost to repair or to take measures for the parts of the roads that have been or is possible to be damaged. For that reason the vulnerability raster maps that the tool produce, around the infrastructure network are not enough. It is needed to go the process a step further and see the possible danger on the infrastructure network lines.

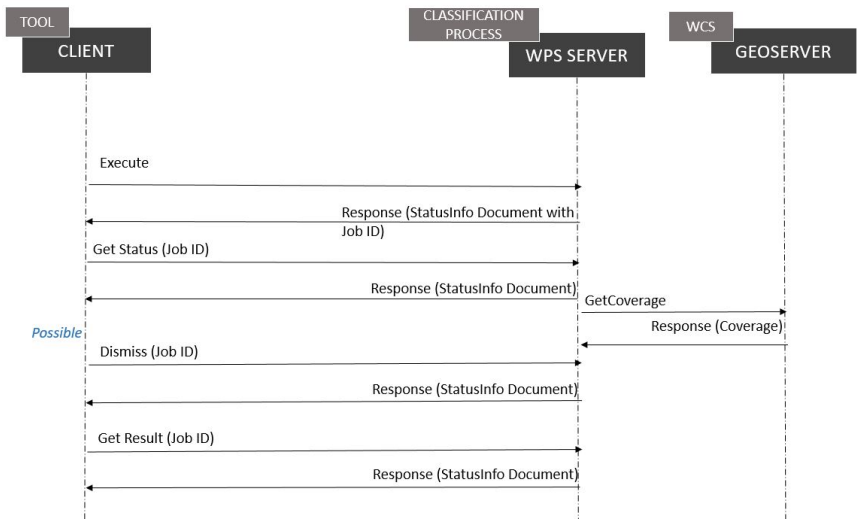

Figure 5.1: Asynchronous execution of the process

In order to develop the translate to risk functionality the steps that should be followed are:

- **Back-end:** Develop a process in the back of the tool for the translate to risk analysis.

In the back end of the tool, a new service should be developed that calculates the average values of the vulnerability raster around the infrastructure network, and attach them as properties in the road lines. The process should be published as a service.

- **Front-end**: Develop the functions to call the process

In order to call the process, the XML request should be created and send to the process. Functions that create and send the response should be developed.

- **Front-end:** Develop the functions to read and display the response.

The response to the process (*GeoJSON* that contains the road lines with properties that indicates the average susceptibility value), should be read and displayed from the client.

- **Front-end:** Reclassify the road lines

  The road lines are displayed with the colors that were sent from the back-end. These colors are result of the classification that happened on the back end. This requirement demand to enable reclassification of the road lines, on the client side.

- **Front-end:** Interface design and development of the translate to risk functionality

The final step is the design and development of an easy to use interface that helps the user to understand how the translate to risk functionality works, and enhance the flexibility of the process.

The translate to risk process has been developed from the main back-end developer of the tool. So the work on this new functionality will be done on the front-end (web browser programming) of the tool.

5.5 EXTRA FUNCTIONALITIES OUTSIDE OF THE SCOPE OF THIS RESEARCH

- Save the project and load it

  In order to produce the final susceptibility map, the user needs to select different parameters through the whole procedure (susceptibility factors' weight, classes, input sources etc.), and sometimes the user wants to save the work with these parameters for later analysis. For that reason, it is important the tool to offer the possibility to save the project and load it.

- Add a new classified layer

  So far the tool offers susceptibility analysis based on the standard susceptibility factors. The motivation for this new functionality was to offer the option to the user to add an extra layer, a classified susceptibility factor, that can be used for the final susceptibility analysis.

6 IMPLEMENTATION

This chapter describes the implementation details, regarding the new functionalities of the RI2DE tool. Decisions that were made and issues that were faced during the development of the new version of the tool will be outlined. The development of the new version of the tool, has been carried out in sequential steps, following the functional requirements of that were described in the previous chapter. At this point it is important to remind that the status reporting and the control of the process as the WPS 2.0.2 standard sets them will not be implemented. Although *PyWPS* implementation of the *GetStatus*, *GetResult* operations is still undergoing, *PyWPS* offers a way of status reporting that is going to be tested to see if it can be applied in the RI2DE tool.

The implementation phase includes the development of codes both at the back-end and the front-end of the tool, tests on the functionality of the new developed version each time, and refinement of the codes if necessary. The structure of this chapter will be: Section 6.1 will briefly describe the setup of the RI2DE environment for development purposes, and will illustrate some examples on the functionality of the back end. Section 6.2 will outline the steps that were followed for the development of the search, retrieve, display, select and use services functionality. Section 6.3 describes an effort that have been made for status reporting. Section 6.4 describes the steps that were followed for the implementation of the translate to risk functionality on the front-end. Section 6.5 describes and illustrates with figures the new GUI and workflow of the user actions. Section 6.6 present the new OGC Web Services Architecture of the tool after adding the CSW.

6.1 SYSTEM SETUP

The first step for every web engineering project is to setup the back-end and front-end environment for development purposes. For that reason:

- The codes of the RI2DE front-end has been downloaded from RI2DE GitHub page and a NodeJs/npm server has been setup in order to run it in development mode. For the debugging of the front-end the Vue-Devtools has been used.
- The WPS of the tool have been downloaded from Deltares SVN repository and a *PyWPS 4.0.0* service implementation that runs on a *python- flask* server has been setup on at the *Linux/Ubuntu* subsystem of Windows.
- A *GeoServer* has been setup, that runs on an *Apache TomCat* server, in order to upload the temporary WMS that are created from the different classification and susceptibility processes of the tool.
- The OSM road and culverts have been loaded to a PostGIS database

The whole setup of the RI2DE environment (both back-end and front-end), was a time consuming procedure. In the Appendix A are described with details all the steps that were followed for the setup of the servers. At this point is should be mentioned that during the setting up of the *PyWPS* environment the biggest issue was created from the *GDAL* dependency. Apparently with some version of the

Linux the newest implementation of *GDAL* is difficult to be installed in a python virtual environment. The problem was solved by installing in *pygdal* library.

For the development of the codes of the front-end, two different branches have been created on RI2DE GitHub page, one for the search, retrieve, display, select and use services functionality, and one for the translate to risk.

6.1.1 Example execute requests to the RI2DE back-end

After the setup of the *PyWPS* server, the next step was to test the classification services in order to understand exactly how they work. As it was already described in the *RI2DE* overview chapter, each classification process accepts as inputs a *roads identifier* and the *layer setup* that contains: *the classes boundaries*, and the *layer name* and *OWS endpoint* of the WCS.

Listing 6.1: XML document used for the execute request for WPS that extracts the roads

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" version="1.0.0" service="
    WPS" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
    schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
      ri2de_calc_roads</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
          geojson_area</ows:Identifier>
      <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
          geojson_area</ows:Title>
      <wps:Data>
        <wps:LiteralData>
        {
          "type": "Feature",
          "properties": {},
          "geometry": {
            "type": "Polygon",
            "coordinates": [ [ [ 4.52306994863682,
                52.059532967349718 ], [ 4.525545891852937,
                52.063131020996131 ],
                [ 4.533252714688852, 52.060605494431172
                    ], [ 4.532523550007451,
                  52.057612733572178 ],
                [ 4.526277420530427, 52.055848121647529
                    ], [ 4.52306994863682,
                  52.059532967349718 ] ] ]
          }
        }
          </wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/json">
      <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
          output_json</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

The *roads identifier* is the output of the WPS that extract the road infrastructures from the *PostGIS* database, and this WPS accepts as input a *GeoJSON* file of the desired polygon area. For that reason, a sample polygon area has been created in the Netherlands through *QGIS*, and has been used as input to this WPS. The execute request to the service and the raw data output of it (GeoJSON with the road lines), are depicted in the Listing 6.1 and Listing 6.2 respectively.

In the next phase, the slope classification process has been selected to be tested. In the execute request there were given as inputs:

- *road identifier* to extract the road lines
- *classes: [0,2,5,90]* , classes boundaries for slope in degrees
- *layer name:* Global_BaseMap:merit_gebco , name of the WCS on the Geoserver
- *OWSurl:* https://fast.openearth.eu/geoserver/ows?, ows url of the Geoserver that the WCS is published

Listing 6.2: Raw output of the WPS that extract the roads: GeoJSON of the road lines

```
{"roadsCollection":
   {"type": "MultiLineString",
   "coordinates":
      [[[4.5293859, 52.0610152], [4.5294783, 52.0610048],
      [4.5295721, 52.0610085], [4.5296616, 52.0610259],
      [4.5297417, 52.0610561]], [[4.5296224, 52.061468],
      [4.5294929, 52.06148], [4.5293643, 52.0614647],
      [4.5292514, 52.0614239]], [[4.5291451, 52.0613355],
      [4.5291181, 52.0612764], [4.5291166, 52.061215],
      [4.5291407, 52.0611555], [4.5291888, 52.0611017]],
      [[4.5292514, 52.0614239], [4.5291898, 52.0613835],
      [4.5291451, 52.0613355]]]},
   "roadsIdentifier": "roads_1568474493573512"}
```

Listing 6.3: XML document used for the execute request for slope classification WPS

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" version="1.0.0" service="
    WPS" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
    schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
   <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
       ri2de_calc_slope</ows:Identifier>
   <wps:DataInputs>
      <wps:Input>
         <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
             roads_identifier</ows:Identifier>
         <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
             roads_identifier</ows:Title>
         <wps:Data>
            <wps:LiteralData>roads_1568474493573512</wps:LiteralData>
         </wps:Data>
      </wps:Input>
      <wps:Input>
         <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
             layers_setup</ows:Identifier>
         <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
             layers_setup</ows:Title>
         <wps:Data>
            <wps:LiteralData>
               {
                  "classes": [0,2,5,90],
                  "layername": "Global_Base_Maps:merit_gebco",
                  "owsurl": "https://fast.openearth.eu/geoserver/ows?"
               }
             </wps:LiteralData>
         </wps:Data>
      </wps:Input>
   </wps:DataInputs>
   <wps:ResponseForm>
      <wps:RawDataOutput mimeType="application/json">
         <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
             output_json</ows:Identifier>
      </wps:RawDataOutput>
   </wps:ResponseForm>
</wps:Execute>
```

In Listing 6.3 and in Listing 6.4 are presented the XML request and response to the slope classification service.

Listing 6.4: Raw Data Output of the slope classification WPS

```
{"style": "ri2de",
 "layerName": "TEMP:slope_1568476349913629",
  "baseUrl": "http://localhost:8080/geoserver/wms"
 }
```

6.2 SEARCH, RETRIEVE, DISPLAY, SELECT AND USE SERVICES

This section of the report describes in detail the different steps and decisions that were made in order to create a tool that can discover and retrieve services from distributed WCS and for a specific susceptibility factor, display them for evaluation purposes, as a "market" of services, and let the user to select one of them as input to the classification analysis, and then use it.

The overview of the classification processes that the RI2DE tool offers showed that the only processes that are flexible enough to accept other datasets, than the default, are the *slope* and the *distance to water* classification. So from an early stage of the implementation the first restriction was set, that the new functionality can be offered only from the slope and the distance to water classification.

Having as case studies the slope and the distance to water classification, the first step was to discover different national, regional, and institutional geoportals that offer WCS, in order to use them for testing purposes.

6.2.1 Geoportals and services

The selection of the geoportals was a difficult task, as the geoportals that are going to be used need to have the following characteristics:

- Support the CSW standard
- Have registered DEM and water surface datasets as WCS, with the *layer name* and the OWS URL of the GIS server they are published written in their metadata.

It was easy to find geoportals that implement the CSW binding protocol, especially in the eu, as the countries are obliged to follow the guidelines of INSPRE directive, for open and interoperable National SDI, but it was not easy to find geoportals with water surface datasets. For that reason the slope classification process became the only case study for testing this new functionality.

While the discovery of Geoportals with elevation WCS was an easier task, as there are many National SDI and institutions that offer elevation datasets, like the *PDOK* Geoportal (National SDI of the Netherlands) that has the *AHN* series of DEM, the most of them didn't have registered in their metadata, the *layer name* and the OWS URL of the GIS server they are published.

Since the knowledge of the *layer name* and the OWS URL is essential for the retrieval of the datasets, it was decided to create a geoportal for the needs of this research, and populate it with records, in order to create an ideal environment for the implementation of the *GetRecords* operation. In a later stage of the project the Deltares *MI-SAFE Data* portal has been populated with records of DEM WCS that the institution used in several projects.

In the Table 6.1 are presented the Geoportals that have been used as test cases, together with their CSW endpoints. In the next sections, follows a brief description of the MI-SAFE Geoportal and on the Geoportal that was created for the purposes of this research.

Table 6.1: Geoportals

GeoPortal	CSW endpoint
MI-SAFE GeoNetwork	https://fast.openearth.eu/geonetwork/srv/eng/csw?
RI2DE (Localhost)	http://localhost:8080/geonetwork/srv/eng/csw?

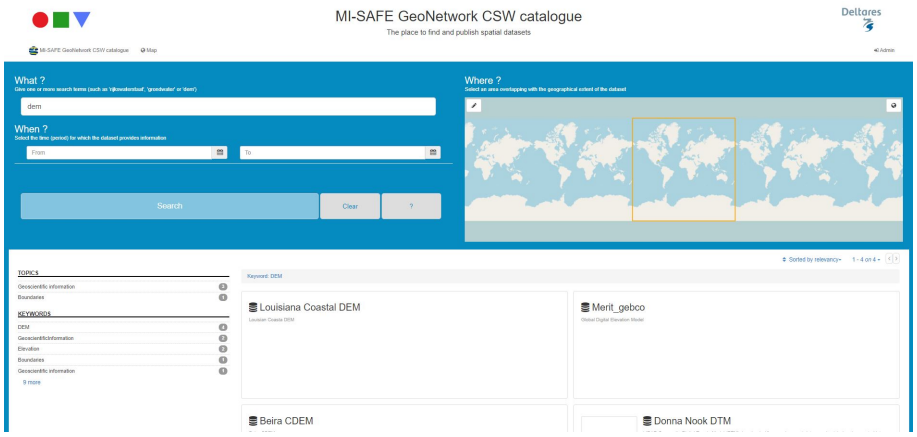

Figure 6.1: MI-SAFE Catalogue

MI-SAFE GeoNetwork

The *MI-SAFE* geoportal is one of Deltares data portals and is has been developed under the FAST project and the *OpenEarth* initiative. In the FAST *OpenEarth* project, a software and web-viewer has been developed that depicts the estimated contribution of coastal vegetation to wave height attenuation for user-selected coastal locations anywhere in the world (publicwiki.deltares.nl). For the purposes of the project a *GeoServer* has been created, for the publishing of the datasets that the tool use. In the *GeoServer* there are DEM of (see also Section 4.3.1)

- Louisiana
- Beira
- Merit_Gebco
- SRTM

When this research started the *MI-SAFE* geoportal had registered only the *SRTM* DEM in its catalogue. For the purpose of this research the Louisiana Coastal DEM, the Beira DEM and the Merit_Gebco has been registered in the catalogue. The Merit_Gebco is the transnational DEM that the RI2DE use for the slope classification process.

Localhost GeoPortal

As it was outlined previously, it was decided to create a geoportal for the needs of this research. The geoportal needs to implement the CSW protocol binding. For the development of the geoportal the *GeoNetwork* open source solution for the development of catalogue application has been used. The name that was given to the new geoportal is RI2DE Catalogue and is not in production. Within the *GeoNetwork* it is possible to harvers metadata records from other *Geoportals* using the harvest operation of the CSW standard or to add new records. Both of the options have been used.

More specifically with the harvest operation the MI-SAFE Geoportal records have been retrieved, in order to alter them by adding the *layer name* and *OWS URL* that are needed.

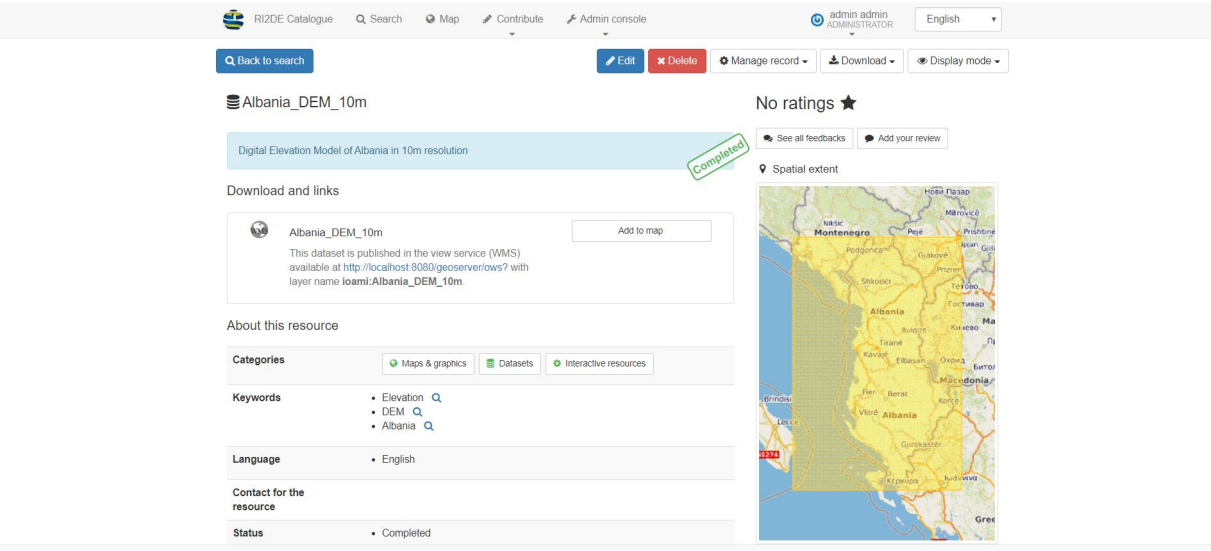

Figure 6.2: RI2DE Catalogue, Albania DEM

Also new records have been added in order to assess the way that records are created and which is the best metadata schema. One of these records was the metadata of an Albania DEM , that was used in many cases as example in the whole development stage.

ALBANIA DEM WEB COVERAGE SERVICE: When the RI2DE tool project started, the first idea was to develop it having as case study the Albania area. For that reason a DEM of the Albania with 10 m resolution and in the national coordinate system (European Petroleum Survey Group (EPSG) 32634) has been used. But when the idea for a regional case study has been abandoned and transnational datasets have been used in order to have global coverage, the Albania DEM has been replaced with the transnational DEM (Merit_Gebco) that is on WGS84.

Since the basic idea of this research, it to discover and use regional datasets with better resolution, the Albania DEM was a perfect case study. For that reason the DEM was published as WCS, in the *GeoServer* that was developed for the purpose of this research, and added as a new record in the RI2DE Catalogue, under the ISO metadata schema.

6.2.2 GetRecords implementation

At this point of the research, the system has been setup for development (both back-end and front-end), and two catalogue services (MI-SAFE Catalogue and RI2DE Catalogue) have been populated with metadata records of elevation WCS for testing purposes. From now on the functional requirements, as they have been defined in the Chapter 5 are going to be the baseline for the implementation of the search, retrieve, display, select and use other services functionality.

The first step according to the list of requirements is to implement the *GetRecords* operation of the CSW binding protocol, in order to "Search" and "Retrieve" the records. Implementing the GetRecords operation means deciding what values are going to be used to the parameters of the request, and also to decide how the request is going to be send.

The first idea was to create in the client side a base XML GetRecords request, and adapt the parameters (e.g. query) according to the case. This request then would be sent directly to the different catalogue services and the client would receive the response with the records that fulfil the query.

This idea was abandoned from an early stage, since it is against to the whole idea for sharing and interoperability, because the whole processing (reading the records and extract the necessary information) would happen on the client side (web browser). For that reason it was decided to use the OWSLib python package. OWSLib is a package of classes and methods that implement the different OGC Web Services operations, serving as a client that sends, receive and process requests. More specifically in the case of the CSW standard, OWSLib offers the *CatalogueServiceWeb* class with methods for every CSW operation. One of this method is the *getrecords (parameters)* that implement the *GetRecords* operation.

Taking advantage this package it was possible to develop a process that sends the *GetRequest* request with the HTTP-POST method, accepts it, reads it, process the returned records, and return only the information that is essential for this new functionality:

- Title of the record
- Abstract of the records
- OWS URL to connect and get it
- Layer name to recognize it

This process has been developed on a way to be flexible enough on accepting different inputs (e.g. it can search either for elevation or water datasets) and has been published under the WPS standard in order to be accessible to anyone.

The first step to develop this process is to implement the ”search” and ”retrieve” part, which means to construct the *GetRecords* operation. From the CSW standard it is known that a *GetRecords* operation should have the following parameters (See also Table 2.4):

- Name of the service
- Version of the service
- Output format
- Output XML schema to be returned
- Maximum number of records to be returned
- The type names to query against
- Element Set Name (”full”, ”summary”, ”brief”
- Query language
- Query

In order to implement the *GetRecords* operation within the OWSlib library the steps are:

INITIATE A *cataloguesserviceweb* OBJECT: An OWSlib *CatalogueServiceWeb* object can be initiated with the parameters that are presented in the Table 6.2. The URL parameter indicates the endpoint of the CSW to connect and send the request. It is essential for the flexibility of the process to have it as a variable, in order to be able to set different URL. The version of the service was left to the default value(”2.0.2”) since the most of the catalogues implement this version of the standard. The skip capabilities parameter was set to ”true” in order to avoid errors that were created during the tests. The rest of the parameters were left either empty or to the default values.

Table 6.2: Initation of CatalogueServiceWeb object

All possible parameters	Values that were given
url	It will be an input to the process. The default that was given is the url of the FastOpenEarth
language (the default is 'en-US'	It was left the default
version (default is '2.0.2')	It was left the default
timeout	It was left the default
skip capabilities	It was set to "true"
username for HTTP basic authentication	It was left empty
password for HTTP basic authentication	It was left empty
authentication	It was left empty

CONSTRUCT THE QUERY: The most important step of the *GetRecords* implementation is to construct the query predicate. The query should be constructed in a way to be flexible to work for every susceptibility factor and to offer also spatial searching. The spatial searching indicates that only datasets that from the selected area (polygon) will be retrieved.

OWSLib package support the OGC Filter Encoding 1.1.0. This Filter encoding offers Logical, Comparison, Spatial and Temporal operators (See Section 2.7.1), and can query against the specified Core queryable properties that every catalogue support.

In order to filter the results according to the area, the*BBOX* spatial operator was used that query the *BoundingBox* property. While in order to search for specific datasets the *PropertyIsLike* operator was used that query against the *AnyText* queryable property, which means that any text value in the metadata records is going to be searched, and if they are like the defined queryable values the datasets are going to be returned. The conceptual structure of the query is depicted in the Listing 6.5. The metadata records that are going to be returned should: Fell in the defined bounding box and have in their texts one of the defined keywords. There in no limit on the keywords to be used.

Listing 6.5: Conceptual structure of query in the OGC Filter Encoding

```
def_handler(self,request,response):
<Filter>
    <And>
        <BBOX>
            <PropertyName>BoundingBox</PropertyName>
                <gml:Envelope>
                    <gml:lowerCorner>Xmin Ymin</gml:lowerCorner>
                    <gml:upperCorner>Xmax Ymax</gml:upperCorner>
                </gml:Envelope>
        </BBOX>
        <Or>
            <PropertyIsLike>
                <PropertyName>AnyText</PropertyName>
                <Literal>Keyword 1</Literal>
            </PropertyIsLike>
            <PropertyIsLike>
                <PropertyName>AnyText</PropertyName>
                <Literal>Keyword 2</Literal>
            </PropertyIsLike>
        </Or>
    </And>
</Filter>
```

CALL THE METHOD GETRECORDS: The final step for the *GetRecords* implementation, is to call the *getrecords* method of the *CatalogueServiceWeb* object that has been already initiated. In this method the parameters and the values that were given are

Table 6.3: Parameters of *getrecords* method

All possible parameters	**Values that were given**
constraints (filter expression)	The query that was created in the previous step
sort by	It was left empty
typenames (default csw:record)	It was left the default
esn (ElementSetName)	It was set to "full" in order to have all the possible metadata information
outputschema (default is: http://www.opengis.net/cat/csw/2.0.2	It was set to http://www.isotc211.org/2005/gmd
outputformat (default is xml)	It was left the default
start position	It was left empty
maxrecords	It was set to 50
xml (raw xml request)	It was left the default
result type (default is 'results')	It was left the default

presented in the. Table 6.3. The *esn* parameter, that is the *SetElementParameter* of the *GetRecords*, was set to "full", because the OWS URL and the *layername* metadata properties, are optional and not included in the shorter version ("brief" and "summary") versions of the metadata. The output schema was decided to set to the "ISO", since the most of the Catalogue Services that were examined, including also the Deltares data portals, support only the "ISO" metadata schema and not the "Dublin Core".

Now that the "search" and "retrieve" part of the process has been developed with the use of the OWSlib package, it is now time to process the returned records in order to retrieve the information that is needed. In the next section is described all the processing that is made in order to extract the title, the abstract, the OWS URL and the layer name of the records, and is explained how the process has been published.

6.2.3 Web Processing Service for the discovery and retrieve of the records from distributed catalogue services

This section will describe how the "search" and "retrieve" that was implemented with the OWSLib package, and the processing of the returned records in order to gain the information that is needed, have been combined and wrapped in one process that is published under the WPS standard. The inputs to this process will be the *Identifier of the road*, *a list of keywords*, and *a list of CSW endpoints*.

As it was already described in the previous section, the records that will be returned should meet the criteria that the query sets, which means to fell in the specified bounding box and to have one of the keywords in their texts (e.g. abstract, title).

The process reads the *road identifier* in order to extract the road lines from the temporary folder and to create the bounding box, and together with the *keywords* creates the query. At the same time the process reads the list with the CSW endpoints and with it every time creates a *CatalogueServiceWeb* object, connecting that way to the Catalogue Service. Then with the query creates and sends the *GetRecords* request, as it was described in the previous section.

The response each time contains the records that fulfilled the query. These records are checked to see:

- If they haven't already recovered from another Catalogue Service
- If they contain all the necessary information (Title, abstract, OWS URL and layer name)

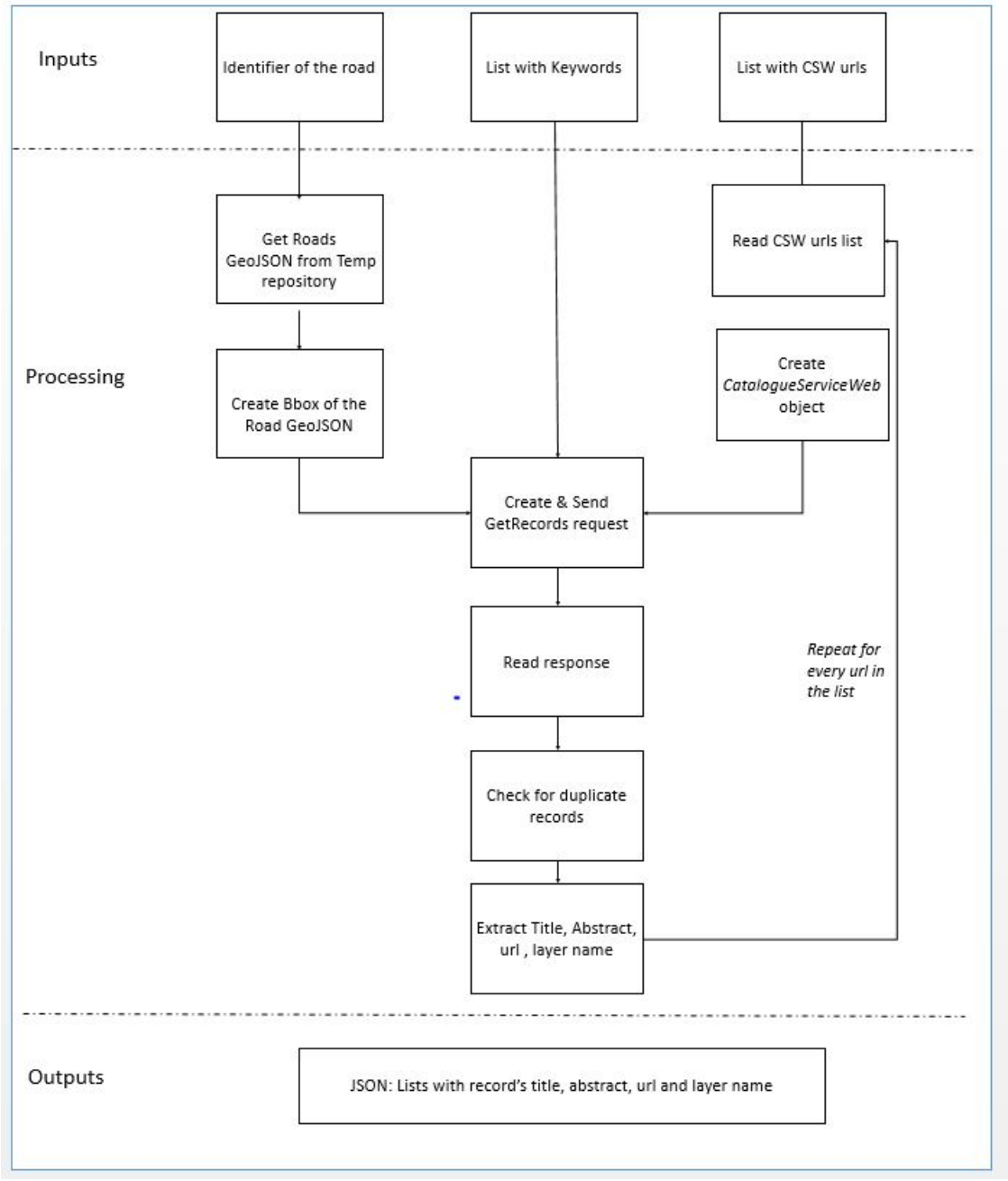

Figure 6.3: Process for search, retrieve and return records from distributed Catalogue Services (WPS)

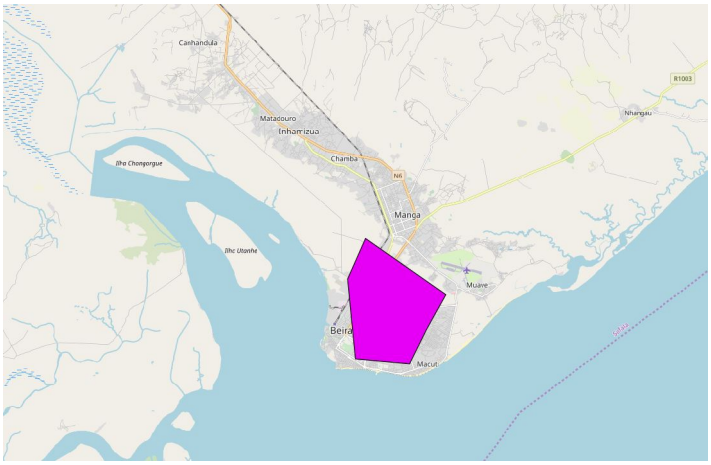

Figure 6.4: Polygon that was created in Beira city of Mozambique, with QGIS

If they do contain it, then they are stored in a list and at the end, all the records that were retrieved from the distributed catalogue services are written in the output. A schematic view of this process is depicted in the Figure 6.3

At this point it is essential to give an example of how this process works. Knowing that in the FAST OpenEarth GeoNetwork there is a record of the Beira DEM WCS, a GeoJSON has been created in the Beira city trough *QGIS*, and has been given as input to the WPS that extract the road lines. After the road lines have been extracted and stored in the temporary folder, the *road identifier* is used in order to call the WPS that discover and retrieve records from distributed catalogue services.

The inputs that were given are:

- Roads identifier that was provided from the WPS that selects the roads
- Keywords: 'dem','elevation'
- CSW endpoint of the MI-SAFE GeoNetwork and the RI2DE (localhost)

The execute request to the WPS is depicted in the Listing 6.6, and the response (JSON that contains the records) in the Listing 6.7. From this response we can see that the process returned global coverage DEM, and the one of Beira, as it was expected.

Listing 6.6: XML document of the Execute request of the WPS that discovers and retrieve the records from distributed catalogue services

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wps:Execute xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
       http://www.w3.org/2001/XMLSchema-instance" version="1.0.0" service="
       WPS" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
       schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
3    <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
         getrecords_url</ows:Identifier>
4    <wps:DataInputs>
5      <wps:Input>
6        <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
             roads_identifier</ows:Identifier>
7        <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
             roads_identifier</ows:Title>
8        <wps:Data>
9          <wps:LiteralData>roads_1568722427210504</wps:LiteralData>
10       </wps:Data>
11     </wps:Input>
12     <wps:Input>
13       <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
             csw_url</ows:Identifier>
14       <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">csw_url</
             ows:Title>
```

```
15          <wps:Data>
16             <wps:LiteralData>{
17             "csw":["http://localhost:8080/geonetwork/srv/eng/csw?" , "
                   https://fast.openearth.eu/geonetwork/srv/eng/csw?"]
18             }
19             </wps:LiteralData>
20          </wps:Data>
21       </wps:Input>
22       <wps:Input>
23          <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
               keywords</ows:Identifier>
24          <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">keywords<
               /ows:Title>
25          <wps:Data>
26             <wps:LiteralData>
27                {
28                   "keywords": ["dem", "elevation"]
29                }
30              </wps:LiteralData>
31          </wps:Data>
32       </wps:Input>
33    </wps:DataInputs>
34    <wps:ResponseForm>
35       <wps:RawDataOutput mimeType="application/json">
36          <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
               output_json</ows:Identifier>
37       </wps:RawDataOutput>
38    </wps:ResponseForm>
39 </wps:Execute>
```

Listing 6.7: Raw data output: JSON that contains the records

```
1  [{"abstract": "Global Digital Elevation Model
2   (DEM) derived from the SRTM (Shuttle Radar Topography Mission).
3   The data is stored on the Deltares P:drive.
4   The SRTM elevation data can also be downloaded
5   from the website http://gdex.cr.usgs.gov/gdex/.
6   This global dataset is freely available, you only
7   need to register first. There is a demo with instructions on
8   the download procedure (http://gdex.cr.usgs.gov/demo/demo.html).",
9   "layername": "global:srtmplus15",
10  "owsurl": "https://deltaresdata.openearth.eu/geoserver/ows?",
11  "title": "global:srtmplus15"},
12  {"abstract": "Digital Elevation Model Merit Gebco",
13  "layername": "Global_Base_Maps:merit_gebco",
14  "owsurl": "https://fast.openearth.eu/geoserver/ows?",
15  "title": "Global_Base_Maps:merit_gebco"},
16  {"abstract": "Beira CDEM",
17  "layername": "Global_Base_Maps:Beira_CDEM",
18  "owsurl": "https://fast.openearth.eu/geoserver/ows?",
19  "title": "Global_Base_Maps:BeiraCDEM"},
20  ]
```

6.2.4 Client processing

Since now it has been implemented the "Discovery" and "Retrieve" part of this new functionality. Following the list of requirements the next step is to work on the client side (web browser) in order to construct the XML requests to the process, develop the equations that display the records and change the source of the classification process, and all these in a new friendly in use GUI.

The XML document that is going to be sent to the WPS server from the web browser, needs the *keywords* of the susceptibility factor, *the CSW URLs* and the *road identifier*. As it was already mentioned in the Section 3.4 the web browser of the tool has been developed with the Vuex *state management* library, which makes easier in this case the manipulation of the datasets, as it is feasible to access them from all the

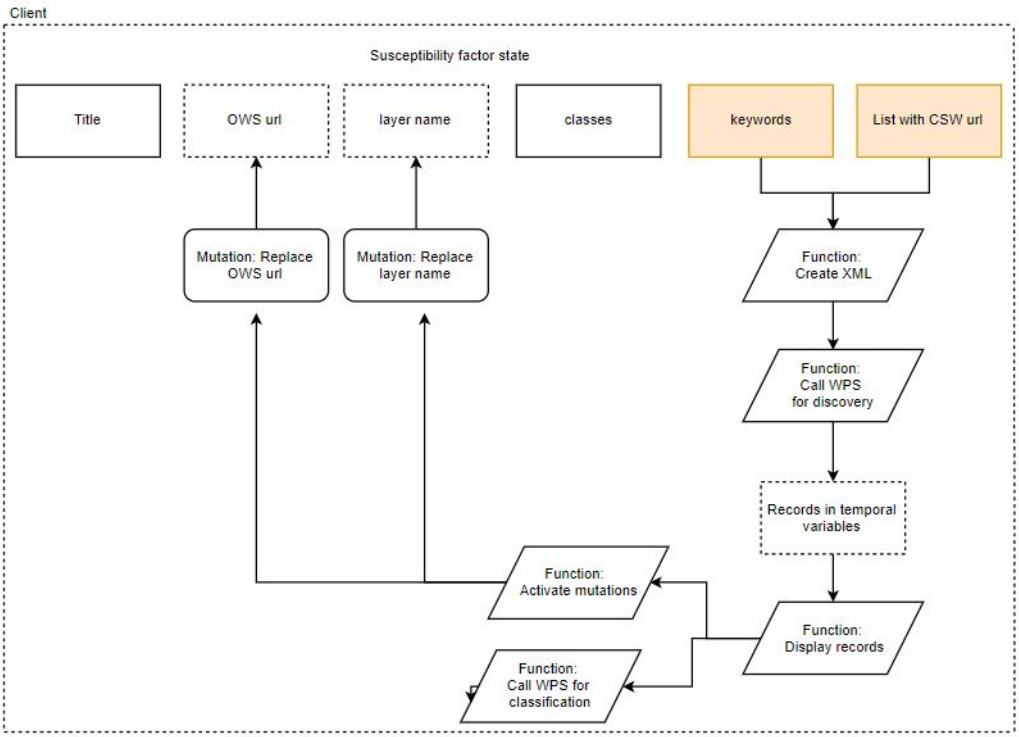

Figure 6.5: Schematic view of the client actions to display and change the source

components of the client and to alter the *state* of them when we want with the *mutations*.The philosophy of the first edition of the tool was to store all the information that configuration JSON file sends to the web browser in *states*, and to create the XML requests to the different WPS by accessing them. Following the same philosophy the keywords and the CSW URLs are stored in two new *states* that were created for them.

As the states of the keywords and the CSW URLs are populated, it is feasible to create the XML document and send the Execute request to the WPS, in order to find and retrieve records of services from distributed catalogues. So when the user choose from the GUI to see extra sources for the selected susceptibility factor, the functions that create the XML and call the WPS are triggered. Then the response to this request, which means the records that were discovered, are stored in a temporary variable and displayed to the user. If the user decides to select one of these datasets, then are activated two functions, one that *mutates* the *states* that stores the *OWS URL* and the *layer name* of the WCS, from the default values that were given from the JSON configuration files, and sends immediately a new Execute request with the new WCS endpoint (that retrieves from the altered states) to the WPS of the classification process. In the Figure 6.5 it is depicted a schematic view of the processing that happens in the client side (web browser). The new GUI is presented in the Section 6.5.

6.2.5 Reprojection

The last requirement in order to complete the implementation of the discover, retrieve, display, select and use other services functionality, is to develop new functions in the back- end of the tool, that reproject the selected sources always to WGS84 coordinate system.

So far the custom procedure for the susceptibility analysis, is to extract the different coverages from the GeoServer that they are published as acwcs, classify them, and then overlay them with the other classified layers in order to produce the final susceptibility map. The susceptibility analysis demands the datasets to be on the

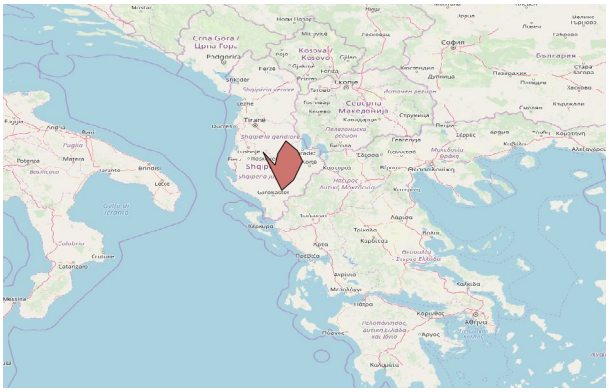

Figure 6.6: Selected area in Albania

same projection. So far the default datasets, were transnational datasets projected on the WGS84 system, so there was no need for a reprojection procedure in the back end. Now that the user can select datasets from regional servers, there is always the possibility to have datasets from different projection systems.

Having a closer look to the custom classification process, it seems that the coverage is extracted with a bounding box that is on the WGS84. For that reason two reprojection needs to happen. One of the bounding box before the *GetCoverage* implementation, in order to be able to extract the coverage, and one after the coverage is downloaded, in order to reproject it to the WGS84 for the final overlay. In the first reprojection, in order to know the correct projection of the WCS, when the connection to the GeoServer is initiated, it is possible to ask for the projection of the coverage service. If the projection then is not the WGS84, then the bounding box should be reprojected to the local system.

Although this procedure can solve problems that occur due to different projections, the double reprojection takes time, especially the reprojection of the coverage, and for a web tool this is not preferred. So the code has not been used, and it can be considered as future work. But the problem still remains, so in order to use other datasets apart from the default transnational ones, they need to be projected in the WGS84 first before they used as inputs to the classification processes.

6.3 TEST OF STATUS REPORTING

As it was already outlined in the PyWPS section of the technology framework chapter of this report, the latest PyWPS version does not still support the GetStatus and GetResult operation of the WPS 2.0.0. For that reason the functional requirements that were described in the Section 5.2 is not feasible to be developed in this framework. Although it is not possible to have job monitor through these operations, PyWPS offers a way for status reporting, similar to the WPS 1.0.0 approach for status report where the process writes the status to a third party service (see Section 3.1.1).

It was decided then to test if it is feasible to get status reporting with the way that the PyWPS offers. Since the status reporting makes sense only when the process takes longer time to complete, it was necessary to select a big area and use as input to the process, a dataset with high resolution. The Albania DEM has a high resolution equal to 10m, so it is a good case study. Moreover since the idea for coordinate transformation inside the classification process was abandoned, the Albania DEM had to be transformed from the national coordinate system (EPSG:32634) to the WGS84 (EPSG:4326) in order to be used as input to the process.

Following the methodology that PyWPS describes for enabling asynchronous execution of the process, the steps for the status reporting implementation were:

STEP 1: Create a polygon (GeoJSON) in the Albania through *QGIS*, big enough to justify the need for status reporting (Figure 6.6)

STEP 2: Execute the WPS that extract the road lines, with input the GeoJSON of the selected polygon, in order to create the roads file and take the roads identifier.

STEP 3: Add to the configuration file of the PyWPS, the output URL to store the status XML document each time (Listing 6.8)

Listing 6.8: Configuration file of PyWPS

```
[server]
...
outputurl=http://localhost:5000/ouputs
outputpath=outputs
...
```

STEP 4: Set the *status supported* and *store supported* parameters in the *def_init* of the slope classification WPS to *true*, (Listing 6.9)

Listing 6.9: Enable store and status support in slope classification WPS

```
def __init__(self):
    ...
        ...
        store_supported=True,
        status_supported=True
        ...
```

STEP 5: Set in the *def_handler* method of the slope classification WPS the status report messages with the *WPSResponse.update_status()* function. In contrast to the buffer example that was given in the Section 3.1.1, where the status report messages are updated every time a feature is buffered, in all the RI2DE classification processes, the processing happens in different parts and is not a repeated procedure (see Custom classification process:). For that reason the slope classification process was executed multiple times, in order to *estimate approximately* which part of the whole processing takes more time to complete, (Listing 6.10).

Listing 6.10: Set status report messages in slope classification WPS

```
def_handler(self,request,response):
...

    response.update_status('Read Input', 5%)
...
    response.update_status('Get Roads', 40%)
...
    response.update_status( 'Converted to slope', 70%)
...
    response.update_status('Classified', 80%)
...
    response.update_status('Applied Mask', 90%)
...
    response.update_status('Uploaded to GeoServer', 100%)
...
```

STEP 6: Send an XML document as Execute request to the WPS server, indicating in the XML that the *output format* should be a *ResponseDocument*, in order to achieve asynchronous execution (Listing 6.11).

After the execution of the slope classification started a response was sent to the client indicating that the process has been accepted and the location to find the

status report. In Listing 6.12, Listing 6.13 and Listing 6.14 are presented the different status report documents. Firstly the status is "Process Accepted" and the percentage done of the process is still 0, which means that it hasn't started yet to *read the input*.Then while the process runs the status changes to "Process Started and the percentage is 40 percent, which indicates that the input has been read and the road lines have been extracted from the PostGIS database. And at the the last document the status changes to "Process succeeded" and the output of the process is provided.

At this test the process execution finished fast enough to not have time to get a status report indicating the 70, 80 and 90 per cent of the process. Which means that perhaps the estimation to set the 40 per cent after the extraction of the road lines is wrong. It is not easy to correctly estimate at which point the process will go faster or slower, for instance when the Internet connection is weak the downloading of the coverage from the GeoServer is slower. So another way for status reporting should be found, perhaps to get the status of each sub-process separately and add it to total calculation. Nevertheless the the idea to have status reporting at the RI2DE tool while this research was happening, have been abandoned until a more accurate way is found to calculate and not just estimate the time that every part of the whole process takes to complete.

Listing 6.11: Asynchronous Execute request of slope classification process

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:Execute xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:xsi="
    http://www.w3.org/2001/XMLSchema-instance" version="1.0.0" service="
    WPS" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://
    schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
      ri2de_calc_slope</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
          roads_identifier</ows:Identifier>
      <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
          roads_identifier</ows:Title>
      <wps:Data>
        <wps:LiteralData>roads_medium_albania</wps:LiteralData>
      </wps:Data>
    </wps:Input>
    <wps:Input>
      <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
          layers_setup</ows:Identifier>
      <ows:Title xmlns:ows="http://www.opengis.net/ows/1.1">
          layers_setup</ows:Title>
      <wps:Data>
        <wps:LiteralData>
          {
            "classes": [0,2,5,90],
            "layername": "ioami:Albania_DEM_10m_global",
            "owsurl": "http://10.0.2.2:8080/geoserver/ows?"
          }
        </wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
   <wps:ResponseDocument status="true" storeExecuteResponse="true">
    <wps:Output >
     <ows:Identifier xmlns:ows="http://www.opengis.net/ows/1.1">
         output_json</ows:Identifier>
    </wps:Output>
   </wps:ResponseDocument>

  </wps:ResponseForm>
</wps:Execute>
```

Listing 6.12: Response Document with status: "Process Accepted and 0 % percent completed

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ExecuteResponse 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 ../
    wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-
    US" serviceInstance="http://127.0.0.1:5000/wps?request=
    GetCapabilities&amp;amp;service=WPS" statusLocation="http:
    //127.0.0.1:5000/outputs/6245680c-d857-11e9-b635-0800279afocf.xml">
    <wps:Process wps:processVersion="1.0">
        <ows:Identifier>ri2de_calc_slope</ows:Identifier>
        <ows:Title>backend process for the RI2DE tool project,
            calculates slope and classifies into classes</ows:Title>
        <ows:Abstract>It uses gdal tools to calculate slope and
            sends back a JSON reply wrapped in the xml/wps format with
            the wmslayer to show</ows:Abstract>
    </wps:Process>
    <wps:Status creationTime="2019-09-16T09:55:43Z">
        <wps:ProcessAccepted percentCompleted="0">PyWPS Process
            ri2de_calc_slope accepted</wps:ProcessAccepted>
    </wps:Status>
```

Listing 6.13: Response Document with status: "Process Started and 40% percentd completed

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ExecuteResponse 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 ../
    wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-
    US" serviceInstance="http://127.0.0.1:5000/wps?request=
    GetCapabilities&amp;amp;service=WPS" statusLocation="http:
    //127.0.0.1:5000/outputs/6245680c-d857-11e9-b635-0800279afocf.xml">
    <wps:Process wps:processVersion="1.0">
        <ows:Identifier>ri2de_calc_slope</ows:Identifier>
        <ows:Title>backend process for the RI2DE tool project,
            calculates slope and classifies into classes</ows:Title>
        <ows:Abstract>It uses gdal tools to calculate slope and
            sends back a JSON reply wrapped in the xml/wps format with
            the wmslayer to show</ows:Abstract>
    </wps:Process>
    <wps:Status creationTime="2019-09-16T09:55:43Z">
        <wps:ProcessStarted percentCompleted="40">b</wps:ProcessStarted>
    </wps:Status>
</wps:ExecuteResponse>
```

Listing 6.14: Response Document with status: "Process Succeeded and output result

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ExecuteResponse 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 ../
    wpsExecute_response.xsd" service="WPS" version="1.0.0" xml:lang="en-
    US" serviceInstance="http://127.0.0.1:5000/wps?request=
    GetCapabilities&amp;amp;service=WPS" statusLocation="http:
    //127.0.0.1:5000/outputs/6245680c-d857-11e9-b635-0800279afocf.xml">
    <wps:Process wps:processVersion="1.0">
        <ows:Identifier>ri2de_calc_slope</ows:Identifier>
        <ows:Title>backend process for the RI2DE tool project,
            calculates slope and classifies into classes</ows:Title>
        <ows:Abstract>It uses gdal tools to calculate slope and
            sends back a JSON reply wrapped in the xml/wps format with
            the wmslayer to show</ows:Abstract>
    </wps:Process>
    <wps:Status creationTime="2019-09-16T09:56:09Z">
        <wps:ProcessSucceeded>PyWPS Process backend process for the
            RI2DE tool project, calculates slope and classifies into
            classes finished</wps:ProcessSucceeded>
```

```
10      </wps:Status>
11      <wps:ProcessOutputs>
12          <wps:Output>
13                  <ows:Identifier>output_json</ows:Identifier>
14                  <ows:Title>Ri2DE calculation of a slope layer [0,1,2]</
                        ows:Title>
15                  <ows:Abstract/>
16              <wps:Data>
17                          <wps:ComplexData mimeType="application/json"
                                encoding="" schema=""><![CDATA[{"style": "ri2de
                                ", "layerName": "TEMP:slope_1568620557442375", "
                                baseUrl": "http://10.0.2.2:8080/geoserver/wms"}
                                ]]></wps:ComplexData>
18              </wps:Data>
19          </wps:Output>
20      </wps:ProcessOutputs>
21  </wps:ExecuteResponse>
```

6.4 TRANSLATE TO RISK FUNCTIONALITY

The need for cost benefit analysis for preventive measures and repair constructions, in case of geohazard risk on specific road segments, motivated the need for a translate to risk functionality . With this functionality the tool calculates the mean values of the raster cells that surround the different road lines, and pass them as properties to the lines.

For that purpose a new process has been developed on the back end of the tool, that translate the produced vulnerability maps to classified road lines. The development of this process has been done from the main back-end developer of the tool. This process is published as a service.

The purpose of this research was to work on the client side of the tool, in order to create a friendly user interface, and create the functions that send the request, and receives and display the classified road lines. The implementation steps are based on the requirements list that was described in the requirements chapter. But before the description of the way the front-end part of the translate-to-risk has been implemented, it is essential to outline the way the translate-to-risk process works.

6.4.1 Translate to risk back-end

In the back end of the tool, it has been added a new process that calculates the mean values of the vulnerability raster cells around the road lines, and create a GeoJSON that includes a feature collection (the road lines) with attached properties that indicate (the mean, max and min value of the possible vulnerability around the line). The GIS analysis is happening with a zonal statistic procedure, so it is needed to know the size of the zone around the road, and also the length that segments are going to be created.

The inputs to this process are:

- Final susceptibility layer setup: The output of the Susceptibility analysis process
- Buffer distance: The distance to create buffer around the road lines. It is needed for the *zonal statistics analysis*
- Segment length: The length to create segment lines (split the lines), in order for the user to configure the accuracy of the calculation (e.g. cut them in smaller pieces than the default ones).
- Road identifier: In order to get the GeoJSON file that contains the road lines, from the temporal directory.

And the output product of this process is a GeoJSON file that contains a feature collection of the different road segments, with properties that indicates the mean, max and min vulnerability value of the segment, and the default color (it is a result of the default classification that is happening in the process).

A schematic view of the translate to risk process, is depicted in the Figure 6.7. The process has been developed in a way to be flexible to the user needs. The user is the one to set the size of the area around the roads (buffer distance) that the risk analysis will be based on, and the length of the road lines.

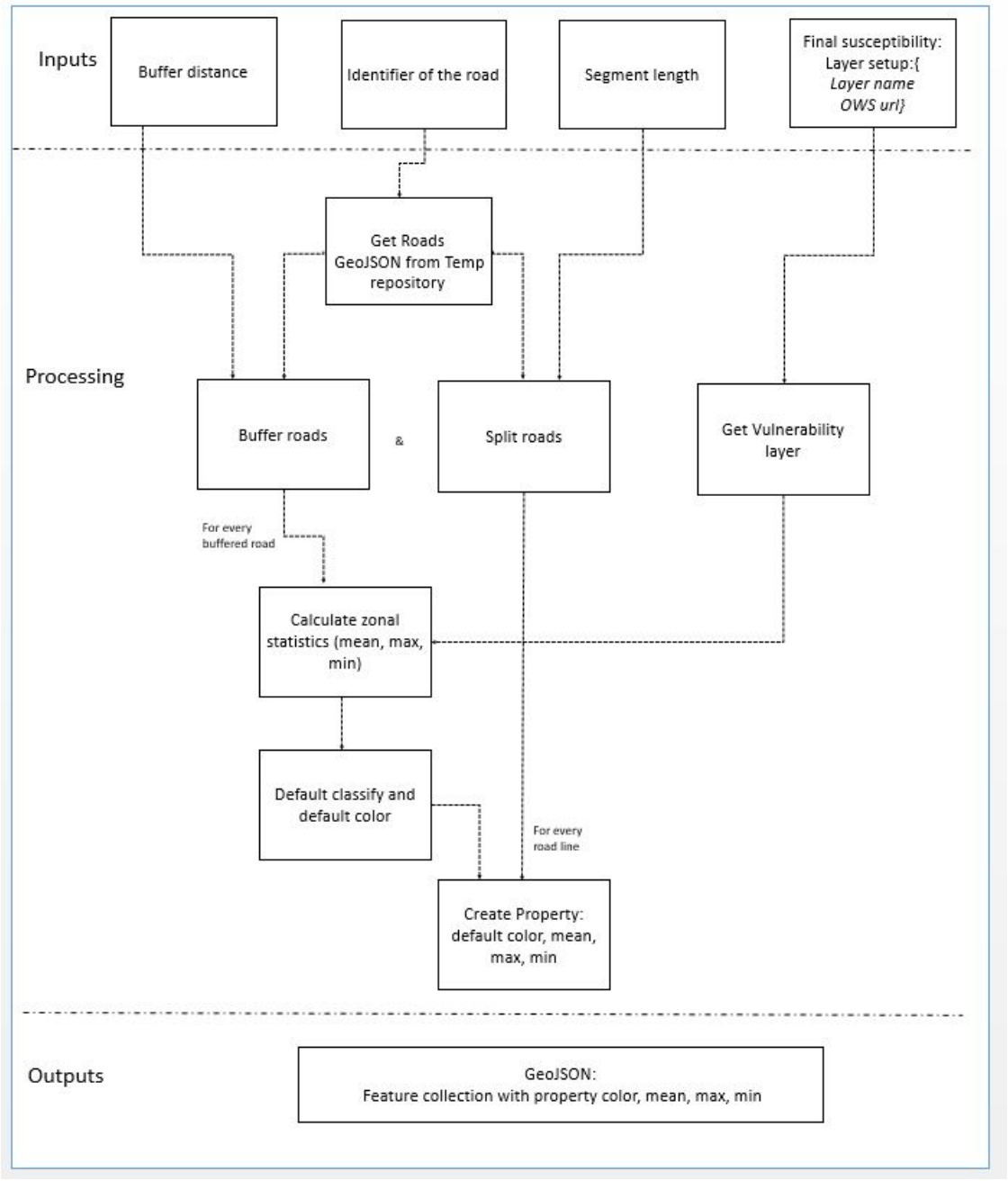

Figure 6.7: Schematic view of the Translate to risk process

6.4.2 Front-end: translate to risk

Having as guideline the requirements list that was described in the requirements chapter, the first step is to develop a function in the front-end that creates the XML

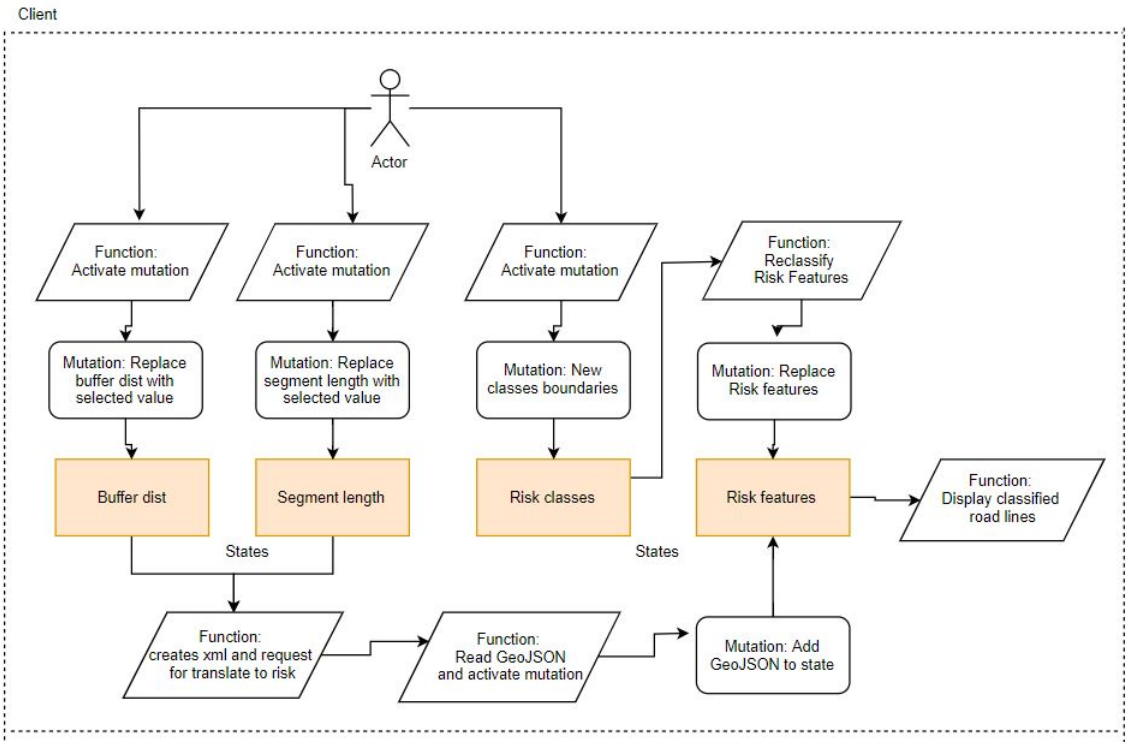

Figure 6.8: Client processing: Translate to risk

request and sends it to the WPS server. This XML request has to contain, the road identifier, the layer setup (l*layer name and OWS URL*) of the temporary vulnerability map, and the buffer distance and segment length. In order to provide a translate to risk functionality configurable to the user needs, there were created two *states* (one for the buffer distance and one for the segments length) that are configurable from the RI2DE GUI. The original values to the states were set to 120 meters for the buffer, and 1000 meters for the segment length, and when the user changes these default values, functions call the *mutations* that change the values of the states. The range to these values was set: 50 to 250 meters for the buffer distance, and 500 to 2000 meters for the segment length.

Having all the necessary information set, the XML request is created and sent to the WPS server. Then a function takes the response to the request, reads the GeoJSON file, that contains the road lines with the necessary properties, and add it to the risk features *state*, that have been created for that purpose. At the same time a function takes the GeoJSON file and add it to the map, with the default colors, as they were provided from the back-end process.

The last of the requirements is to enable an on the fly reclassification in the front end. In order to accomplish that, a *state* has been created, that was called risk classes, on which the boundaries of the classes are going to be stored. The boundaries were set to be from 0 to 2, and they are configurable from the front end (web browser). When the user change them, a function *mutates* them. When the risk classes boundaries change, automatically a function is called that *mutate* the *state* of the risk features (change the color in the feature properties, according to the new classes boundaries). Then the new risk features are displayed in the map.

A schematic view of the processing that happens on the front-end, from the creation of theXML request, till the reclassification of the risk features, is presented in the Figure 6.8.

6.5 NEW GUI AND WORKFLOW OF USER ACTIONS

In the Section 4.4.7 it has been presented the workflow of the user actions at the first version of the RI2DE too. After the implementation of the new functionalities, discover extra sources for the slope susceptibility factor and translate to risk functionality, the workflow has changed. In this section the new workflow will be described, emphasizing mainly on the new functionalities. The old functionalities will be only referred. This workflow will contain also the functionalities that were implemented by the RI2DE developers team. Figure 6.9, Figure 6.10, Figure 6.11 illustrate the RI2DE GUI.

1. *Load project or create a new one*

   In the previous workflow, when the first page of the RI2DE was loaded, the start screen with the global map was presented. Now when the **ro2de!** (ro2de!) is loaded for first time, the users firstly have to select if they are going to load a project or create a new one.

2. *Selection of the hazard*

   Remains the same as the previous workflow

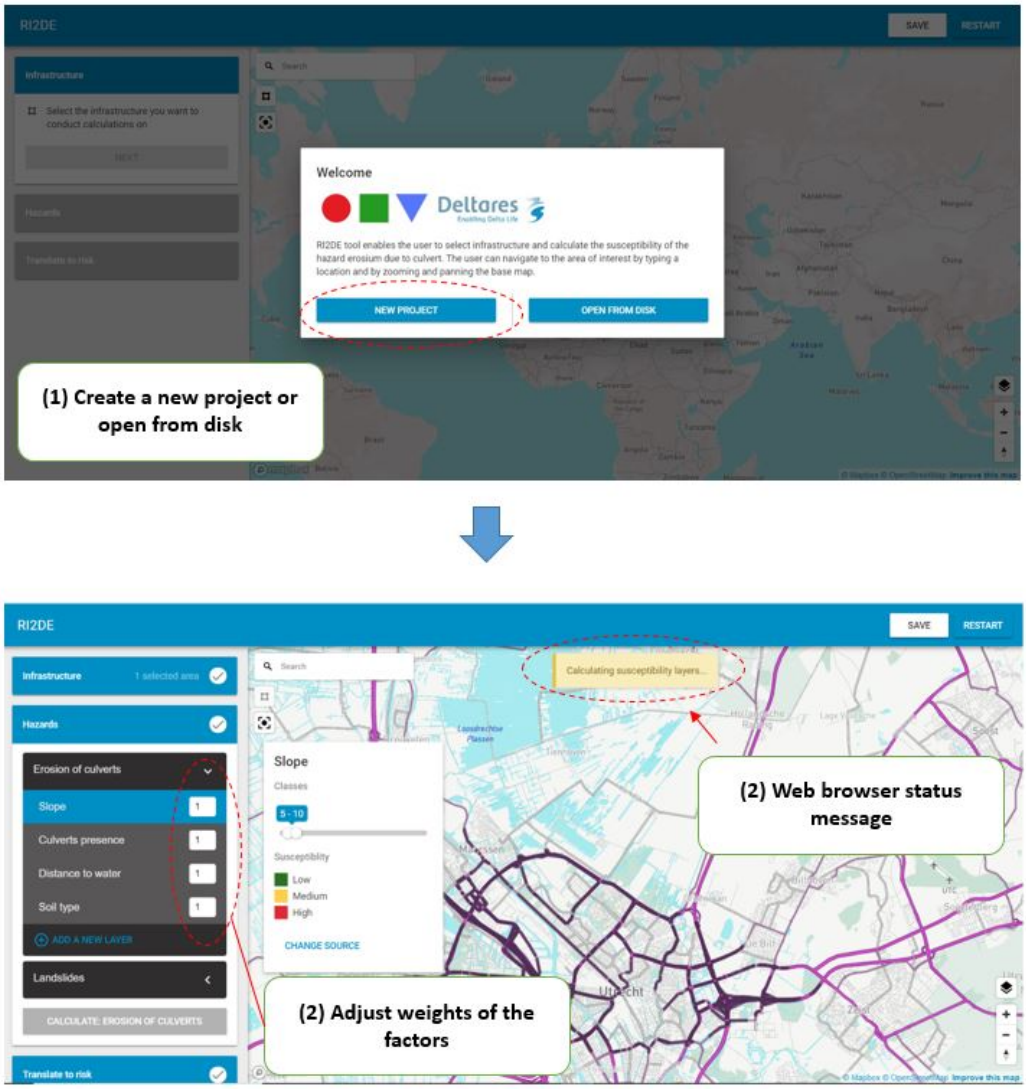

Figure 6.9: RI2DE GUI, part 1

3. *Selection of the infrastructures*

   Remains the same as the previous workflow

4. *Configure the susceptibility factors*

   In the window of the slope susceptibility factor, were the user can adjust the boundaries of the classes, there is a new button, the change source. When the user clicks the change source button, a window pops up, displaying the available DEM WCS on the area, as they were retrieved from the *GetRecords* operation. In the list with the services, the users can see the title and the abstract, and if they want, can select them as inputs to the slope classification process.

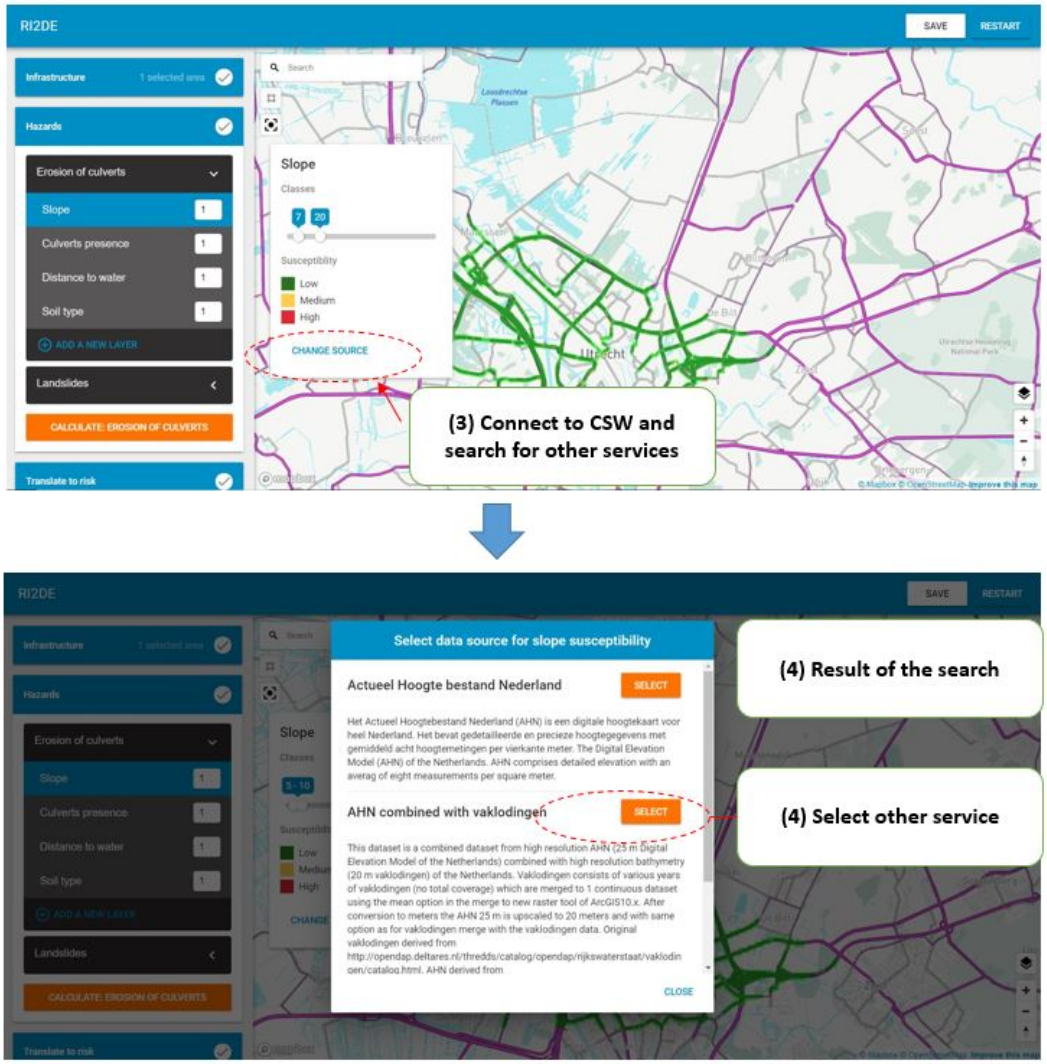

Figure 6.10: RI2DE GUI, part 2

5. *Calculate Susceptibility map*

   When the users set the parameters of the susceptibility factors, the calculate button can be pressed, as before, in order to calculate the final vulnerability map of the selected hazard. But in this version, the vulnerability map calculation is not the last step.

   When the calculate button is clicked then the tool is navigated to the translate to risk page.

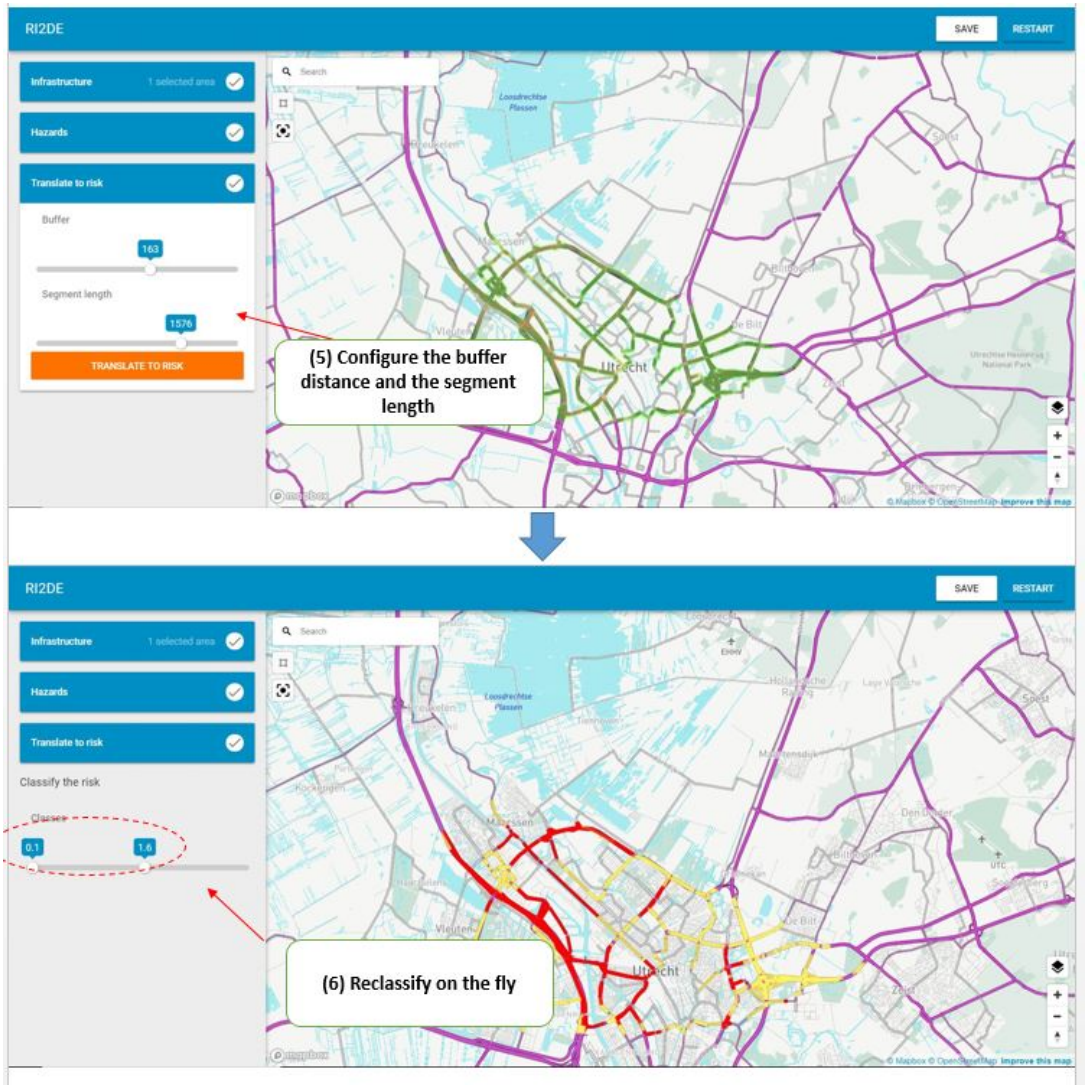

Figure 6.11: RI2DE GUI, part 3

6. *Configure translate to risk parameters*

   In this page now the users can alter the default values of the buffer dist and the segment length, by sliding the respective bars. When the user adjust the parameters to the desired values the translate to risk button can be pressed, and the translate to risk calculation is initiated.

7. *On the fly reclassification of the classfified road segments*

   When the translate to risk process ends, the classified road lines are displayed. Then the user can adjust the boundaries of the classes, in a sliding bar that has appeared on the left of the screen. Every time the user plays with the boundaries, the road features are reclassified on the fly.

In the second snapshot of the Figure 6.9 the web browser shows a "calculating susceptibility layers" status message while it waits form the WPS server the response. This is a good indication of how it is feasible to have status messages out of the WPS framework.

6.6 NEW OGC ARCHITECTURE

The discovery of new source functionality, introduced a new component in the OGC Web Services Architecture of the tool, the OGC CSW. An extra level has been added to the different levels of the architecture, which includes Distributed Catalogue Services that implement the CSW binding protocol. In the level that represents the WPS, a new process has been added, the Discover & Retrieve records process, that connects directly to the different catalogues (with the operation) in order to extract metadata for DEMs. For that reason, the basic levels of the Distributed repositories and the Distributed GIS Servers, have been expanded, on containing extra DEMs and services from the default ones. A schematic representation of the new OGC Web Service Architecture of the tool is presented in the Figure 6.12. The new components are emphasized with yellow compared to the parts of the old architecture.

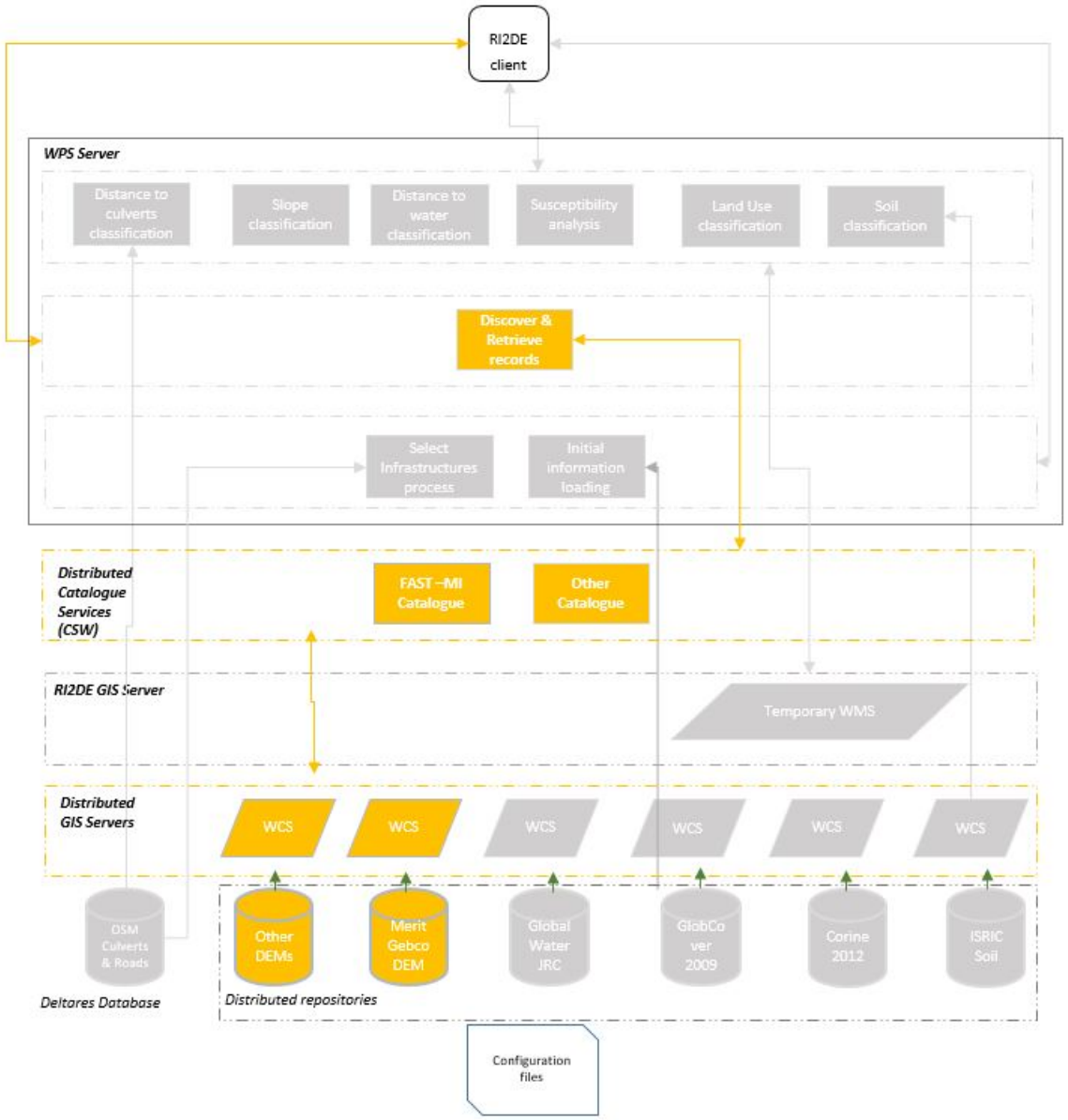

Figure 6.12: New OGC Web Services Architecture of RI2DE

7 DISCUSSION AND CONCLUSIONS

In this chapter a discussion will be made and conclusions will be given, on what this research tried to achieve and on the final product which is a new version of the RI2DE GIS web tool.

The main purpose of this research was to evaluate if the standards, their implementations and the organizations that produce spatial datasets and tools are mature enough to support the development of GIS web applications that demands up-to-date regional datasets, and complex geo-processes, for quality decision making. Motivation and case study the same time for this research, has been the RI2DE GIS web tool of Deltares institution, a tool that that is solely developed from open sources technology components and with OGC standards. By setting a set of functionalities that address the needs of the users, and what the standards offers, a research was carried out through the effort to implement a new version of the tool. The new functionalities that were tried to be developed were:

1. The users will be able to discover and retrieve distributed services that are registered on regional or national SDIs and select them as the inputs of the vulnerability processes.
2. Status report message, e.g in the form of a progress bar will be sent to the user while the processes are running.
3. The users will be able to control/stop the process if it takes long time to complete.
4. There will be a new section on the tool where the user can translate the vulnerability raster maps into classified road lines based on the vulnerability of the surrounded area.

These functionalities were tried to be developed under the context of researching the:

- The CSW standard that is responsible for publishing standardized catalogues (geoportals).
- The WPS 2.0.2 that offers the specifications for asynchronous execution of the processes, and the operations to control and monitor the jobs. Which is important for long time running processes.
- The free and open source PyWPS 4.0.0 implementation of the standard under which the different geoprocesses the tool offers are published.

With the delivery of this thesis two of the four desired new functionalities, have been implemented, under the existing technology framework (PyWPS limitations at supporting the new operation of the standard). Nevertheless the discover and retrieve services from distributed SDIs was a success, and a big achievement, since not many projects exist that integrate in one web platform GIS analysis (WPS implementation) and discovery of services focused on this analysis (CSW implementation). Moreover the translate of the raster vulnerability maps into classified road lines has been also implemented. While simple into its implementation, the last functionality can be a long time running process, which makes it ideal for future work on the job control and job monitor of the process.

The functionality that makes possible to discover and retrieve services from SDIs is already online at the https://ri2de.openearth.eu. While the translate to risk functionality is currently available only as a deploy preview on the GitHub page of the tool https://deploy-preview-51--ri2de.netlify.com/.

7.1 CONCLUSIONS ON THE RESEARCH QUESTIONS

In order to server the main purpose of this research firstly the sub- questions and the main research question should be answered.

1. *How flexible are the Web Processing Services of the tool?*

   The classification processes of the tool are published as WPS, which means that they have a standard structure of inputs and outputs. Every classification process has been developed according to a custom classification procedure, and the structure of the inputs that concerns the datasets has to be a JSON layer setup that contains information regarding, the classes boundaries, and the *OWS URL* and *layer name* of the WCS that offers the dataset.

   This question is trying to answer how easy is to change the values of these inputs. Answering the first sub-question it will make clear if the discovery of new data sources for the classification processes makes sense. The flexibility of the classification process on accepting new datasets is based on the way they have been constructed. If the classification process is highly based on the input dataset, then there is no meaning to discover new sources. The answer to this question was that only the slope classification WPS can accept other WCS from the default ones.

   A good overview of the way each classification process had been developed follows.

   DISTANCE TO WATER CLASSIFICATION PROCESS: The Web Processing Service that offers classification for water datasets, according to their distance to the surface water, can be considered flexible enough to accept inputs from any source. The only issue is that the classify function needs the resolution of the input dataset in order to apply the final classification, and this resolution is hardcoded to 25 meters, equal to the default water dataset (Global Surface Water JRC). In order for this classification process to be considered flexible enough to accept also other inputs from the default one, the resolution values should be extracted from the dataset that is inserted each time.

   SOIL CLASSIFICATION PROCESS: The Web Processing Service that offers classification of the soil datasets, is not quite flexible to accept other inputs than the default ones. The classify function has been constructed based on the soil types of the ISRIC soil datasets. For that reason the layer setup, in the execute request, is always empty, since it is not possible to alter the classes, or the input coverage service. Only if we provide as input a dataset that has the exact soil types and seven depth layers as the ISRIC datasets, the process will work.

   LAND USE CLASSIFICATION PROCESS: Similar to the soil classification process, the land use classification process is not flexible in accepting other inputs than the default ones. The classify function has been constructed based on the land use codes of the Corine 2012 land use coverage for the Europe, and the GlobCover 2009 coverage for the rest of the world.

CULVERTS CLASSIFICATION PROCESS: The Web Processing Service that offers classification of the culverts, has been constructed in a different logic that the rest of the processes. The dataset that the classification is based on is the OSM culverts lines, which are firstly transformed to raster, and then classified. In the layer setup of the execute request the OWS URL and the layer name are always empty, but the classes boundaries are needed. So while the process has configurable classes boundaries, is not flexible on accepting other datasets. The process can be considered flexible only if we provide another dataset of culverts lines, and change the culvert lines name from fixed inside the code to configurable.

SLOPE CLASSIFICATION PROCESS: The Web Processing Service that offers classification of elevation datasets, according to the slope, is flexible in accepting other inputs DEMs that the default one (MertiGebco DEM). Within the layer setup are provided, the classes boundaries, and the OWS URL and layer name of the elevation Web Coverage Services. With the OWS URL and the layer name the coverage is downloaded from the GeoServer, then the elevation datasets are transformed into slope, and base on the provided classes the classification happens. For that reason and because the elevation datasets are easy to find, the slope classification became the case study for the new functionality that enables the discovery of other services than the default one, and set them as inputs to the classification process.

2. *How to get the metadata of the Web Services from the distributed Catalogues?*

   Answering the second sub-question outlined the way to retrieve the OWS URL and the layer name of the Web Coverage Services that are registered as metadata records in different OGC Catalogue Services world wide.

   To answer this sub-question a research had been carried out on the OGC Catalogue Services specifications, in order to understand the different rules it sets for the development of catalogues. The research showed that the OGC Catalogue Services specifies an abstract information and interface model that every catalogue can implement through a specific binding protocol. The OGC Catalogue Services for the Web binding protocol is the most common in use. The operation that enable the discovery and retrieve of records from Catalogues that support the CSW, is the GetReocrds.

   The first step for the GetRecords implementation was to select catalogues for testing purposes. The selection was a difficult task as the catalogues should support the CSW binding protocol, and have registered DEM Web Coverage Services. Although it was easy to find catalogues that support the CSW standard and have DEM WCS records, it wasn't easy to find in their metadata the layer name and OWS URL. For that reason the catalogues that had been used were, a catalogue that had been created for the purpose of this research and the Fast OpenEarth Geoportal of Deltares institution, in order to have administrator rights and be able to alter the records.

   For the implementation of the GetRecords request to the geoportals, the OWSLib python library had been used, in order to create a process that search and retrieve the desired metadata properties and publish it later as Web Processing Service, in order to be shared to anyone.

   This process constructs the GetRecords request, which contains a query, in order to discover datasets that fell into a specific bounding box and contain specific keywords, and sends the request to the different catalogues. The records that are returned, had be chosen to be in the ISO metadata schema since it is the most common. From this records the Title, abstract, OWS URL and layer name are extracted.

3. *How to establish the connection between the Catalogue Services and Web Processing Service ?*

   Answering this sub-question it made possible to pass from the OWS URL and layer name properties that were returned from the Catalogue Services search to the slope classification Web Processing Service.

   In order to accomplish that the work had be done to the front-end of the tool, with the state management of the datasets that the VUEX JavaScript framework offers. For that purpose XML request for the Web Processing Service that discover and retrieve the metadata had been constructed, and the returned records from this request were stored to states. One state had been created for the OWS URL and one for the layer name. On that way when the user selects the dataset to use as input to the slope classification process, the states are mutated with the new OWS URL and layer name, and the request to the slope classification process is sent having the new OWS URL and layer name in the layer setup.

4. *What is the status of the PyWPS 4.0.0 with respect to the Web Processing Services 2.0.2 version (test of job monitoring and job control)?*

   The free and open-source implementation of the Web Processing Service, PyWPS, has recently released the PyWPS 4.0.0 version, which has been developed in order to implement the Web Processing Service 2.0.2 version of the standard, and more specifically the GetStatus, GetResult and Dimsiss operation that offers job monitoring and job control of the process respectively.

   PyWPS community decided to re-write the PyWPS code from scratch based on new knowledge and new technologies, but they still have not implement the GetStatus, GetResult and Dismiss operations. A few efforts had been made from volunteers and thesis projects but they are still under development.

   But although the PyWPS does not support the GetStatus, GetResult and Dismiss operations yet, there is a way to get status report messages through the Execute request. An effort had been made to provide status messages from the classification processes of the tool, but the way the processes are constructed does not offer reliable results. PyWPS is not the only project that does not support yet the WPS 2.0.2 version of the standard. From a research that carried out, it was realized that until the delivery of this thesis, only the ZOO-Project supports it.

5. *What will be the GUI and new work flow of the user actions?*

   In order to provide the new functionalities, the sequence of actions that a user of the tool can take had to change. Different designs of the new GUI has been created and refined during the whole research. The final design has been created in a way to offer an easy work flow of actions in order to discover and select new datasets for the slope classification process, and to translate the risk of hazard to the road lines. In the Section 6.5 is described in detail the new work flow of the user actions.

Having answered the different sub questions and having implemented the new functionalities of the tool, the main research question can answered:

"What will be the new OGC Web Services Architecture of the tool"

The discovery and select new services functionality, has altered as it was expected, the OGC Web Services Architecture of the tool. A new level of services, has been introduced, the level that contains the distributed OGC Catalogue Services, and a new process has been added in the WPS server. This process connects with distributed Catalogue Services and extract the records of the Web Coverage Services that can be used as inputs to the classification processes.

These Web Coverage Services are distributed word wide in different GeoServers and are accessible through their OWS URL and layer name. The lack of flexibility of the most of the processes, restricted the discovery and select functionality only to the slope classification process. For that reason in the level of repositories and services, there are the default datasets and services for the rest of the susceptibility factors, apart from the slope susceptibility factor, which it is possible to contain any DEM dataset and service.

7.2 CONTRIBUTION TO THE FIELD OF GEOMATICS

Working during this thesis on a demanding GIS web tool of an organization, that is based on open source technology components and only on standard data formats, it made possible to explore the existing standards and the state-of-the-art technology framework for their implementation, from the three aspects of this research: standards, implementation and organizations. One of the achievements of this research and contribution on the same time of at the field of Geomatics, is the realization that these three levels (standards, implementation and organization) are not in line. While the standards are there and initiative like INSPIRE set the rules for standardized SDIs, its not easy to discover datasets or services, and the metadata are poor.

The greatest achievement and contribution of this thesis to the field of Geomatics, is the integration of the CSW and WPS standard in one architecture of decision making tool. Not any similar implementation has been discovered during the implementation of this thesis. There are SDIs that offer searching on registered and harvested metadata of processes, and services not in a similar way.

The searching that the WPS that was created takes place in distributed catalogues servers, and is focused on datasets that the processes of the platform needs. The retrieved records are filtered and only the ones that have accessible services are displayed. While the services are not registered correctly and the processes are not flexible enough (it can be solved with replacement of the processes with new ones), the tool works.

7.3 DISCUSSION

At this section a more general discussion will be made concerning the realizations from the achievements and limitations of this research.

7.3.1 Discover, retrieve, display, select and use services from distributed catalogues

The greatest achievement of this research was the development of one platform which offers distributed searching of Web Coverage Services, in different OGC Catalogues world wide and use of these services as inputs to the susceptibility analysis Web Processing Services. This achievement was carried out with a lot of restrictions regarding the flexibility of the processes to accept new inputs and the discovery of available Web Coverage Services.

Working on the implementation of this functionality made me realize that although a big amount of spatial datasets is being produced in a daily bases, and although they organization publish them, this is not enough. The metadata are needed in order to discover them. And the metadata should be populated on the correct way. One of the organization that populates all the metadata with the URL and the layer name of the services is the ISRIC organization. It is not enough to find them, but it is needed to access them also.

The ”metadata is amazing” phrase can encapsulate the biggest limitation of this research.

7.3.2 Job monitor and job control

The initial ambition when this research started was to develop a tool that can monitor the status of the process and offer to the user the possibility to stop them when they take time to complete. These functionalities has not been implemented during this research. The PyWPS implementation of the WPS 2.0.2 version of the standard is still undergoing. But it is not the only project that has not implement it yet, (only the ZOO-Project has a full implementation of the standard) which arise question regarding the complexity of the implementation. The PyWPS community has started the implementation since 2016, and there is an open conversation. The code is free and open, so anyone can help in the implementation.

These functionalities were not implemented due to incompatibility of the PyWPS technology framework with the latest release of the WPS. This outlines perhaps the biggest problem with the free and open-source tools, like the PyWPS. PyWPS development is depended only on volunteers work and on student research projects, which makes it difficult to have continuous updates in order to support the latest versions of the standard.

7.3.3 Translate to risk

During this research for the translate to risk functionality I worked only on the front-end of the tool. A general remark that it should be mentioned, is that working in a project that has only standard formats and services as components, made the whole procedure an easy task in the collaboration with the others, and into passing information from the server to the client. Developing processes as WPS and provide them to a front-end developer for further programming in the web browser was pretty easy, as the only thing that the developer needs to know is the inputs and the outputs of the process. During this functionality all it was needed to know was the values that should be passes to the service, and that the output was a GeoJSON.

7.3.4 Evaluation of the tool

Due to time constraints it has not be made a usability test. Nevertheless during this research, the supervisors from TU Delft, the stakeholders from Deltares institution, and the team of developers (including me) have evaluated the tool in many aspects. What is needed to be taken into consideration is that for someone that is not familiar with the steps that should be made and the parameters that need to be configured, the tool does not have a ”self-descriptive” GUI. For example in order to access the discover services functionality, the more sources button should be pressed in the susceptibility factor page. It is not easy for someone to understand what this button does or even what is the difference with the add layer button, that add a classified susceptibility factor.

As regarding the functionality part of the tool, when this research started I had high ambitions as regarding the CSW integration in its architecture. At the end it was implemented, but apart from the limitations that the discovery of datasets sets, the greatest issue is the creation of flexible of classification processes.

7.4 FUTURE WORK

This section present some suggestions for future work either were not implemented due to time or technology limitations.

- **Usability testing**: A questionnaire can be created and answered in an evaluation session from different users, in order to test how self-descriptive is the GUI of the tool. So far the evaluation has been made only from the
- **Job monitor and job control**: The PyWPS implementation of the WPS 2.0.2 is still undergoing, but the PyWPS community has an open conversation on the topic, so perhaps in the near future the implementation will be available. The PyWPS source codes are free and open to anyone so the implementation of the GetStatus, GetResult and Dismiss operation can be a future thesis research. On the other hand there is another alternative, it is possible to implement the job control and job monitor operations at the RI2DE tool with the ZOO-Project.
- **Job monitor at the sub-processes**: Even with the current implementation of the PyWPS the status report is possilbe. The problem was the separation of the classification processing in many sequential steps. A good solution would have been either to get report messages regarding the step that the process runs, or to find a way to calculate each sub process percentage of completion.
- **CRS Transformation**: The CRS transformation functionality has to been implemented as it leads to long time running processing. For that reason in a future work a separate WPS can be created in order to transform first the dataset and the use it as input to the WPS of the classification.
- **Connection to the catalogues**: In the version of the tool that was created, the user can discover new sources from different catalogues, and have a list with title and abstract of the retrieved records. An idea for future work would be to add a button next to the record, that connect directly to the catalogue in order to have a full access to its metadata.
- **More user friendly GUI**: A better GUI perhaps can be achieved by adding e.g a manual or a link to the wiki of the tool with the steps that the user should follow in order to reach to the final product. Also instead of configure some values through the configuration files, e.g the keywords, it can be added a kind of configuration in the web browser

BIBLIOGRAPHY

Baumann, P. (2010). *OGC WCS 2.0 Interface Standard - Core*. Open Geospatial Consortium Inc.

Beaujardiere, J. (2006). Opengis Web Map Server implementation specification. *Open Geospatial Consortium Inc.*, OGC Standards Document 06-042, Version 1.3.0.

Calero, J. S. (2017). MI-SAFE architecture. URL https://publicwiki.deltares.nl/display/OET/MI-SAFE+architecture. [Online; accessed 25 October 2019].

Čepický, J. and de Sousa, L. M. (2016). New implementation of OGC Web Processing Service in Python programming language. PyWPS-4 and issues we are facing with processing of large raster data using OGC WPS. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*.

De Boer, G. (2019). Openearth. URL https://publicwiki.deltares.nl/display/OET/OpenEarth. [Online; accessed 25 October 2019].

de Boer, G. and Smits, B. (2017). MI-SAFE. URL https://publicwiki.deltares.nl/display/OET/MI-SAFE. [Online; accessed 25 October 2019].

Degree (2019). degree Webservices. *OSCGeo Project*, Version 3.4.10.

ESA (2019). GlobCover. URL http://due.esrin.esa.int/page_globcover.php. [Online; accessed 25 October 2019].

ESRI (2019a). Tutorial: Publishing a WPS service. URL https://enterprise.arcgis.com/en/server/latest/publish-services/windows/tutorial-publishing-a-wps-service.htm. [Online; accessed 25 October 2019].

ESRI (2019b). What is ArcGIS Enterprise. URL https://enterprise.arcgis.com/en/get-started/latest/windows/what-is-arcgis-enterprise-.htm. [Online; accessed 25 October 2019].

Falemo, S., Blied, L., and Danielsson, P. (2015). ROADAPT Ooads for today, adapted for tomorrow guideline part c GIS-aided vulnerability assessment for roads. *Conference of European Directors of Roads*.

Gebco (2019). Gridded Bathymetric Data. URL https://www.gebco.net/data_and_products/gridded_bathymetry_data/. [Online; accessed 25 October 2019].

GeoNetwork (2019). Geonetwork opensource. URL https://geonetwork-opensource.org/. [Online; accessed 25 October 2019].

GeoServer (2019a). GeoServer is an open source server for sharing geospatial data. URL http://geoserver.org/. [Online; accessed 25 October 2019].

GeoServer (2019b). WPS operations. URL https://docs.geoserver.org/2.13.2/user/services/wps/operations.html#execute. [Online; accessed 25 October 2019].

GeoServer (2019c). Web Processing Service (WPS). URL https://docs.geoserver.org/2.13.2/user/services/wps/index.html. [Online; accessed 25 October 2019].

GeoServer (2019). WPS Service page. URL https://docs.geoserver.org/latest/en/user/services/wps/administration.html. [Online; accessed 25 October 2019].

Gistandards (2018). GIS Standards. URL https://www.gistandards.eu/gis-standards/. [Online; accessed 25 October 2019].

Gooseon, Y., , Kwangseob, K., and Kiwon, L. (2017). Linkage of OGC WPS 2.0 to the e-government Standard Framework in Korea: An Implementation case for geo-spatial image processing. *International Journal of Geoinformation*.

Hengl, T., de Jesus, J. M., Heuvelink, G. B., Gonzalez, M. R., Kilibarda, M., Blagotić, A., Shangguan, W., Wright, M. N., Geng, X., Bauer-Marschallinger, B., et al. (2017). SoilGrids250m: Global gridded soil information based on machine learning. *PLoS one*.

Hevner, A. and Chatterjee, S. (2010). *Design Research in Information Systems*. Springer.

ISO (2019). Standards.

Kralidis, T. (2019). Owslib 0.18.0 documentation. URL https://geopython.github.io/OWSLib/. [Online; accessed 25 October 2019].

Laza, A. (2018). Process isolation in PyWPS framework. Master's Thesis, Czech Technical University in Prague.

Mapbox (2019). How Mapbox works. URL https://docs.mapbox.com/help/how-mapbox-works/. [Online; accessed 25 October 2019].

Mueller, M. and Pross, B. (2016). *OGC WPS 2.0.2 Interface Standard: Corrigendum 2*.

Nebert, . D., Voges, U., Vretanos, P., Bigagli, L., and Westcott, B. (2016a). *OGC® Catalogue Services 3.0 Specification - HTTP Protocol Binding*.

Nebert, D., Voges, U., and Bigagli, L. (2016b). *OGC® Catalogue Services 3.0 - General Model*.

Nebert, D., Whiteside, A., and Vretanos, P. (2007). Opengis catalogue services specification. *Implementation Specification*.

Nogueras-Iso, J., Zarazaga-Soria, F. J., and Muro-Medrano, P. R. (2005). Geographic information metadata for spatial data infrastructures. *Resources, Interoperability and Information Retrieval*.

52 North (2019a). Innovative open source software. URL https://52north.org/software/software-projects/. [Online; accessed 25 October 2019].

52 North (2019b). Standardized web-based geo-processing. URL https://52north.org/software/software-projects/wps/. [Online; accessed 25 October 2019].

52 North (2019c). wps-js-client. URL https://github.com/52North/wps-js-client. [Online; accessed 25 October 2019].

OGC (2019a). Certified and implementing products. URL https://www.opengeospatial.org/resource/products/compliant. [Online; accessed 25 October 2019].

OGC (2019b). OGC History (abbreviated). URL https://www.opengeospatial.org/ogc/history. [Online; accessed 25 October 2019].

PyWPS (2019a). Extensions. URL https://pywps.readthedocs.io/en/latest/extensions.html. [Online; accessed 25 October 2019].

PyWPS (2019b). Pywps 4.3 documentation. URL https://pywps.readthedocs.io/en/latest/.

Schut, P. (2007). Opengis Web Processing Service. *Open Geospatial Consortium Inc.*, OGC Standards Document 05-007r7, Version 1.0.0.

team, Z.-P. (2019). ZOO-Project documentation. *ZOO*, Release 1.8.

UNC (2019). Metadata for data management: A tutorial.

van den Brink, L. (2018). Geospatial Data on the Web. Phd thesis, Delft University of Technology.

van Koningsveld, M. and den Heijer, K. (2014). Data. URL https://publicwiki.deltares.nl/display/OET/Data. [Online; accessed 25 October 2019].

Vretanos, P. (2010). *OpenGIS Filter Encoding 2.0 Encoding Standard*. Open Geospatial Consortium.

Vretanos, P. A. (2005). Opengis Filter Encoding Implementation Specification. *Open Geospatial Consortium Inc.*, OGC Standards Document 04-095cl, Version 1.1.0.

Vuex (2019). What is Vuex. URL https://vuex.vuejs.org/. [Online; accessed 25 October 2019].

Wikipedia (2019a). GeoJSON. URL https://en.wikipedia.org/wiki/GeoJSON. [Online; accessed 25 October 2019].

Wikipedia (2019b). GeoServer. URL https://en.wikipedia.org/wiki/GeoServer. [Online; accessed 25 October 2019].

Wikipedia (2019c). Openstreetmap. URL https://en.wikipedia.org/wiki/OpenStreetMap. [Online; accessed 25 October 2019].

Wikipedia (2019d). State management. URL https://en.wikipedia.org/wiki/State_management. [Online; accessed 25 October 2019].

Wikipedia (2019e). JAON. URL https://en.wikipedia.org/wiki/JSON. [Online; accessed 25 October 2019].

Wikipedia (2019f). Vue.js. URL https://en.wikipedia.org/wiki/Vue.js. [Online; accessed 25 October 2019].

YAMAZAKI, D. (2018). MERIT DEM: multi-Error-Removed Improved-Terrain DEM. URL http://hydro.iis.u-tokyo.ac.jp/~yamadai/MERIT_DEM. [Online; accessed 25 October 2019].

A SYSTEM SETUP

The RI2DE GIS web tool is developed from only free and open source technology components and has an OGC Web Services Architecture which means that it is reproducible by anyone. At this appendix guidelines will be given on how to setup the back-end and the front-end of the tool.

A.1 SETTING UP PYWPS 4.0.0 IN A LINUX ENVIRONMENT

It is suggested to setup the back-end in a Linux environment, since PyWPS is not tested on MS Windows platforms, which don't support yet the asynchronous execution. RI2DE runs with python 2.7.

Step 1:
Check Python 2.7 installation on the system and make sure the version is up to date. Update and upgrade the system

Step 2:
Create a virtual environment and activate it **Step 3:**

Download RI2DE WPS from RI2DE SVN

Step 4: Install at the virtual environment the dependencies with the *pip install -r requirements.txt*
It should be noted that some Linux versions create issues with the GDAL dependency. The tool needs the GDAL 2.3.2 version and after. The easiest way to install GDAL if something goes wrong, is through the pygdal package.

Step 5: Install cURL at your linux

Step 6: Run the pywps.wsgi. Now the service run on your localhost:5000.

Step 7: Test the services by sending request with the curl command through the terminal. Example: *"curl -X POST -H "Content-Type: application/xml" -d @request.xml http://localhost:5000/wps"*

A.2 SETTING UP RI2DE WEB BROWSER IN A LINUX ENVIRONMENT

It is suggested to setup the front-end of the RI2DE tool in a Linux environment.
Step 1:
Install Node.js and npm

Step 2: Clone the repository from RI2DE GitHub page

Step 3: Change the name of the .env.example to .env and set the Mapbox Token. It is possible to get a Mapbox Token for free from Mapbox.

Step 4: Run the npm install command

Step 5: After everything are installed, run npm dev in order to run it in a development mode

COLOPHON

This document was typeset using LaTeX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from André Miede.