

A Method for Enhancing Shareability and Reproducibility of Geoprocessing Workflows. Case Study: Integration of Crowdsourced Geoinformation, Satellite and In-Situ Data for Water Resource Monitoring

ROBERT OUKO OHURU

March 2019

SUPERVISORS:

Dr. Ir. R.L.G Lemmens

Dr. J.M. Morales

A Method for Enhancing Shareability and Reproducibility of Geoprocessing Workflows.

Case Study: Integration of Crowdsourced Geoinformation, Satellite and In-Situ Data for Water Resource Monitoring

ROBERT OUKO OHURU

Enschede, The Netherlands, March 2019

Thesis submitted to the Faculty of Geo-Information Science and Earth Observation of the University of Twente in partial fulfilment of the requirements for the degree of Master of Science in Geo-information Science and Earth Observation.

Specialization: Geoinformatics

SUPERVISORS:

Dr. Ir. R.L.G Lemmens

Dr. J.M. Morales

THESIS ASSESSMENT BOARD:

Prof. Dr. M.J. Kraak (Chair)]

Dr. S. Jirka (External Examiner, 52°North Initiative for Geospatial Open Source Software GmbH)

DISCLAIMER

This document describes work undertaken as part of a programme of study at the Faculty of Geo-Information Science and Earth Observation of the University of Twente. All views and opinions expressed therein remain the sole responsibility of the author, and do not necessarily represent those of the Faculty.

ABSTRACT

With the recent advancement in technology, a large amount of heterogeneous and distributed geospatial data is becoming available. As a result, scientists are faced with the challenges of integrating these large data in geoprocessing functions to solve complex scientific problems. The rise of web service technology offers an opportunity for processing functions and geospatial data to be shared online in form of web services thereby ensuring interoperability and accessibility of geoprocessing resources. Most scientific solutions require several geoprocessing functions and resources some of which cannot be provided by a single computing resource and therefore calls for distributed processing in the web in what is popularly known as grid computing. Workflows present a framework in which complex geoprocessing functions and geospatial data can be combined and executed automatically in real-time. Integrating geospatial and processes in a workflow has been approached by popular GIS software packages. However, these software packages do not incorporate geoprocessing functions exposed through web services thus making it difficult to create shareable and reproducible workflows.

Several standard organizations have proposed standards that, if implemented, can support shareability and reproducibility of geoprocessing workflows. The implementation of OGC WPS supports interoperability and accessibility of geoprocessing functions while WFS, WCS and SOS provide specifications for sharing geospatial data. WfMC and OMG have also come up with standard notations and schema for modelling and describing workflows such as BPMN and XPD. However, these standards have not been appreciated a lot in current GIS Workflow Management Systems (WfMSs) mainly because they don't represent current technology advancements. For instance, BPMN and XPD are purely XML-based and often support SOAP services which do not align to the current trend for RESTful services and lightweight protocols such as JSON. This, therefore, calls for a more generally accepted standard which borrows from the workflow implementations of current WfMSs. To do this, we propose a method for enhancing the sharing and reproducibility of geospatial workflows which implements two approaches. First, by establishing a standard workflow interchange schema based on a JSON data format. Using this interchange format, we create a method for transforming workflow from one WfMSs to another based on the mapping of their constructs. Secondly, we provide a method for composing workflows from heterogeneously distributed geoprocessing functions using web services. We implement a web-based prototype system to offer a visual abstraction of the underlying method for workflow composition which also has a backend workflow engine responsible for service chaining and workflow execution. We demonstrate the applicability of our method using a simple workflow for triple collocation which combines crowdsourced geoinformation, satellite and in-situ data. The execution of this workflow provides a similar result to the methods used in ILWIS desktop application for triple collocation which supports shareability and reproducibility of the workflow using our method.

Keywords

Shareability, Reproducibility, Geoprocessing Workflow, Web Services Chaining, Triple Collocation.

ACKNOWLEDGMENTS

If it were not for the overwhelming support in form pieces of advice and prayers from my supervisors, family and friends, this thesis would not have been a reality. I want to express my deepest gratitude to all of you.

First, and the most important, I would like to thank the Almighty God for the wisdom, grace, strength and good health that He bestowed upon me to finish this thesis. For sure, my heart was never troubled for I believed in a God who his greater than all my weaknesses.

I want to thank my supervisors, Dr. Ir. R.L.G Lemmens and Dr. J.M. Morales for being the best mentors that I ever needed during my research period. Their guidance and every time spent to review my thesis helped to transform my weakness into strength as a young scientist. I wouldn't have reached this far if it were not for their patience. I am equally grateful to Dr. Ir. Luc Boerboom, Dr. Ir. Bert Toxopeus and Dr. Ir. Chris Mannaerts who also supported me in various ways to ensure the success of my research. They consistently enquired about my progress and encouraged me through this journey.

I am indebted to Bas Retsios for being a friend and a role model in software engineering. I went to his office for every technical problem that I met along the way and he gave me enough time beyond appointments.

My sincere appreciation goes out to my parents Elder John Ohuru and Mrs. Milka Akoth together with my brothers Brian, Jacob and Esau for enduring my absence for such a long time. It was not easy for them, but they provided me with a peace of mind and prayed earnestly for me to finish this thesis in good health.

To my friends Stella, Andy, Eustace, Esther, Dantom, Godfrey and many who I cannot mention, your help was precious. Your kindness, encouragements and prayers offered great help to me throughout this research.

Special thanks to everyone who helped me along the way.

May God bless you!

TABLE OF CONTENTS

1.	Introduction.....	11
1.1.	Background Information	11
1.2.	Problem Statement.....	13
1.3.	Research Objective.....	14
1.4.	Research Questions.....	14
1.5.	Use Case.....	15
1.6.	Thesis Outline	15
2.	Workflows.....	17
2.1.	Evolution of Workflows	17
2.2.	Workflow Modelling.....	18
2.3.	Scientific Workflows	20
2.4.	Factors Affecting the Reproducibility of Workflows.....	22
3.	Workflow Management Systems.....	25
3.1.	Workflow Specification Standards	26
3.2.	Standardization compliant WfMSs.....	28
3.3.	Non-Standardization Compliant WfMSs	29
3.4.	Shortcomings of Current WfMSs.....	32
3.5.	Proposed Solution for the Challenges facing Current WfMSs.....	35
4.	Workflow Composition from Distributed Web Services.....	40
4.1.	Composability of Scientific Workflows.....	41
4.2.	Data Services	42
4.3.	Processing Services	48
4.4.	OGC Process Chaining	54
4.5.	Workflow Engine	55
5.	Supporting Shareability and Reproducibility of Workflows	57
5.1.	Supporting Shareability through Standard Interchange Format	57
5.2.	Provenance Support for Reproducibility	68
5.3.	REST API to Support Reuse	70
6.	Prototype implementation.....	72
6.1.	System Architecture	72
6.2.	Generic Workflow Client.....	75
6.3.	Data Services	77
6.4.	Processing Services	81
6.5.	Workflow Engine	87
6.6.	Workflow Transformation.....	93
7.	Proof of Concept.....	100
7.1.	Satellite, In-situ and Crowdsourced Geoinformation.....	100
7.2.	Triple Collocation.....	102
7.3.	Shareable and Reproducible Workflow for Triple Collocation.....	103
7.4.	Result Discussion	108
8.	Conclusions and Recommendations	111
8.1.	Conclusions	111
8.2.	Limitations	117
8.3.	Suggestions for OGC Standards.....	118
8.4.	Suggestion for GIS Software Developers	118
8.5.	Recommendations for Future Work.....	119

LIST OF FIGURES

Figure 2.1: Basic BPMN elements	18
Figure 2.2: Topological Sorting of Processes using DAG.....	20
Figure 2.3: Comparison of the causes of workflow decay.....	23
Figure 2.4: Comparison of Workflow decay due to third-party resources.....	24
Figure 3.1: Composition of Workflow Management System.....	25
Figure 3.2: The Model Driven Architecture framework.....	37
Figure 3.3: Architecture of Workflow Interchange formats.....	38
Figure 4.1: Levels of Composability of Scientific Workflows.....	41
Figure 4.2: Sensor Web Enablement Framework	46
Figure 4.3: The WfMC Workflow Architecture	55
Figure 5.1: Abstract Class diagram for a Workflow.....	59
Figure 5.2: Class diagram for the Workflow Schema	68
Figure 5.3: Flowchart for process discovery.....	69
Figure 6.1: System Architecture	73
Figure 6.2: The Generic Workflow Client's User Interface.....	76
Figure 6.3: RESTful Service Definition through the Workflow client.....	87
Figure 6.4: Workflow Transformation.....	94
Figure 6.5: Changing resource providers for the same process.....	97
Figure 6.6: Transformation of PIW to QGIS Workflow.....	99
Figure 7.1: Study Area, Dano Burkina Faso.	103
Figure 7.2: Time-series Analysis of Sensor Data.	104
Figure 7.3: Abstract Workflow for the Triple Sensor Water Accounting.....	105
Figure 7.4: Concrete Workflow for Triple Sensor Approach.	106
Figure 7.5: Triple Sensor Workflow Composition from Web Services.....	106
Figure 7.6: JSON extract of the Triple Sensor Workflow.	107
Figure 7.7: Visualization of BPMN-based Triple Sensor Workflow in Camunda modeler.	108
Figure 7.8: Result Analysis of Triple Sensor Workflow Execution.....	109

List of Listings

Listing 4.1: WFS GetCapabilities Response.....	43
Listing 4.2: WFS DescribeFeatureType Response.....	43
Listing 4.3: WCS GetCapabilities Response.....	45
Listing 4.4: WPS GetCapabilities response using SOAP bindings.....	49
Listing 4.5: RESTful WPS GetCapabilities Response	50
Listing 4.6: WPS DescribeProcess using SOAP Bindings.....	50
Listing 4.7: RESTful WPS DescribeProcess Result.....	51
Listing 4.8: WPS Execute Request's Body for SOAP Binding	52
Listing 4.9: WPS Execute Request's Body for RESTful Binding	53
Listing 5.1: JSON schema for workflow metadata property.....	59
Listing 5.2: JSON schema for the properties of an operation	60
Listing 5.3: JSON schema for the operation's metadata property.....	62
Listing 5.4: JSON schema for an operation's input.....	64

Listing 5.5: JSON schema for an operation's output.....	65
Listing 5.6: JSON Schema for connection property.....	67
Listing 6.1: Snippet of the JSON Representation of a Workflow	77
Listing 6.2: Snippet for the Python implementation of WFS GetCapabilities	78
Listing 6.3: JSON object for GetCapabilities request.....	80
Listing 6.4: OGC WPS GetCapabilities response for gs:Centroid operation.....	82
Listing 6.5: XML Snippet for OGC WPS DescribeProcess for gs:Centroid.....	83
Listing 6.6: Code snippet for mapping of WPS process definition to standard JSON schema	83
Listing 6.7: JSON representation for WPS gs:Centroid operation.....	84
Listing 6.8: Sample WPS Execute Body.....	85
Listing 6.9: JSON representation for the AggregateRainfall RESTful service.	87
Listing 6.10: WPS Root element specification.	88
Listing 6.11: Python Code Snippet for WPS Execute Implementation.....	89
Listing 6.12: Python Code for generating URL for RESTful web service.	90
Listing 6.13: Recursive Function for Insertion Sort.....	91
Listing 6.14: Code Snippet for Finding the Execution Order of Operations.....	91
Listing 6.15: Code Snippet for Executing the Workflow.....	92
Listing 6.16: Code snippet for initializing the process element and sequence flows (connections).....	94
Listing 6.17: Code snippet for creating service tasks and data inputs.....	95
Listing 6.18: Code snippet for mapping JSON connections to BPMN serviceFlows.	96
Listing 6.19: An extract of a BPMN sequenceFlow for a simple workflow.	96
Listing 6.20: Code snippet for searching an operation based on a keyword.....	98

LIST OF TABLES

Table 3.1: BPMN Process Elements	27
Table 3.2: BPMN Diagram Elements	27
Table 3.3: OGC WPS Process Elements.....	28
Table 3.4: ILWIS Workflow Elements	30
Table 3.5: QGIS Workflow Elements	32
Table 3.6: Observed Differences among selected WfMSs	33
Table 5.1: Mapping Workflow Elements for different Workflow Specifications.....	58
Table 5.2: RESTful API for Service Reuse	70
Table 6.1: WFS Operations	78
Table 6.2: WCS Operations	80
Table 6.3: SOS Operations	81
Table 6.4: WPS Operations	82
Table 6.5: Non-OGC Compliant RESTful Services.....	86
Table 7.1: Comparison of results from the Triple Sensor Workflow to findings by Mannaerts et al. (2018)	109

LIST OF ABBREVIATIONS

API	Application Program Interface
BPEL	Business Process Execution Language
BPDM	Business process definition metamodel
BPMN	Business Process Modelling Notation
DAG	Directed Acyclic Graph
GPW	Geo-Processing Workflow
ILWIS	Integrated Land and Water Information System
JSON	JavaScript Object Notation
MDA	Model Driven Architecture
OGC	Open Geospatial Consortium
OMG	Object Management Group
PIW	Platform Independent Workflow
PROV	Provenance Model
PSW	Platform Specific Workflow
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
WCS	Web Coverage Service
WfMC	Workflow Management Coalition
WfMS	Workflow Management System
WFS	Web Feature Service
WMS	Web Mapping Service
WPS	Web Processing Service
WSDL	Web Service Definition Language
SDI	Spatial Data Infrastructure
SOA	Service-Oriented Architecture
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
SensorML	Sensor Modelling Language
UML	Unified Modelling Language
VGI	Volunteered Geographic Information
XML	Extensible Markup Language
XPDL	XML Process Definition Language
XSD	XML Schema Definition

1. INTRODUCTION

1.1. Background Information

Remote sensing technology and in-situ measurements observed from local weather stations are the two traditional sources of geospatial data that have extensively contributed to the scientific research. One of the scientific application of data obtained from these sources has been in the management of water resources. For instance, in monitoring the growth of the harmful algae blooms in recreational water bodies and drinking water (Clark et al., 2017), evaluation of extreme precipitations for water resource and flood risk management (Dhib et al., 2017). Better water resource management is critical to helping people, economies, and ecosystems to thrive, reduce poverty and sustain prosperity. However, successful water management requires detailed knowledge of the available water resources which can only be achieved through effective monitoring and forecasting. Water resource monitoring entails the provision of adequate qualitative and quantitative information about the state of the water resource at any moment (Garcia et al., 2016). Getting the latest and specific information for water resource monitoring or disaster management is a challenge with many satellite products and in-situ generated data. This is because of the low temporal and spatial resolution of these data sources.

The last decade has seen the emergence of a third data stream where humans are involved in scientific research by creating and sharing information. When this information generated by humans contains geospatial references, it is known as Volunteered Geographic Information (VGI; Dhib et al., 2017). The term Volunteered Geographic Information (VGI) was first coined by Goodchild (2007) to refer to geospatial data created and disseminated voluntarily by individuals. Literature materials use other terms to describe VGI such as crowdsourcing, citizen science, citizen observation or participatory science (Assumpcao et al., 2018). Their differences notwithstanding, these terms are often used interchangeably to depict the act of involving the public in collection and dissemination of data. Crowdsourced geoinformation suggests a complementary source of data to fill the gaps in satellite and in-situ data.

With the recent advancement in web and mobile phone technology, a large amount of heterogeneous and distributed geospatial data is becoming available. As a result, scientists are faced with the challenges of integrating these data to solve specific problems. To promote the automated integration of these data for solving complex scientific issues, well defined sequential methods contained as a workflow can be used (Yue et al., 2011). Workflow is a concept that has existed in the business domain for an extended period and has been useful in facilitating the automatic execution of business processes. Researchers in various fields have embraced the use of workflows to conduct a range of analysis and scientific pipelines since

they model computation structure and data processing tasks in a manner that help in the management of a scientific process.

Integrating datasets and processes in a workflow has been applied by popular GIS software packages including ESRI suite Model Builder, ILWIS model builder, QGIS processing modeler, ERDAS Imagine Spatial modeler among others. However, these software packages are proprietary and are often confined in a desktop installation making it difficult to share workflows with different users and across different platforms. Furthermore, these workflows are only executable within their propriety software since they depend on a combination of software and libraries contained in the environment of their propriety GIS software. Sharing workflows helps scientist to understand scientific processes created by their colleagues as well as make the workflows as an essential building block in their new processes. Reproducibility of a workflow involves taking the original workflow, data and rerunning the execution to give the same results (Taylor et al., 2007). Reproducibility is very vital in scientific processes to help scientists to validate and verify a given set of results. Reproducibility allows a workflow created for a particular scientific problem to be reused by different users by a repetition of steps with varying sets of data to produce new or more elaborate results. Shareability and reproducibility of workflows are an important application requirement towards achieving interoperability and accessibility of geospatial resources which includes data and processes.

Interoperability can be addressed by establishing common standards, amongst which enabling the accessibility to geospatial resources through web services (Yue et al., 2012). Several organizations have been involved in establishing standards to control access and sharing of geospatial resources. In 1993, the Workflow Management Coalition (WfMC) was created to promote and develop the use of workflows through the establishment of standards for software terminology, interoperability and connectivity among processes (Schmidt, 1999). They developed a large set of reference models, documents and standards with the primary focus on processes. For instance, they came up with the XML process definition language (XPDL) in 1998 as an interchange format for business process models. Its popularity was further enhanced when WfMC endorsed the Business Process and Modelling Notation (BPMN) as a graphical standard for business processes in 2004 (Ko et al., 2009). To this date, XPDL is still being used for describing processes. The Open Geospatial Consortium (OGC) has also specified several standards that can be used to create geoprocessing workflows in an interoperable way by combining processes and data using web services. These standards include Web Processing Service (WPS) which can be built into workflows to execute remote processes that have been exposed by different GIS software. The OGC's Sensor Web Enablement (SWE) has a suite of standards to handle spatial data infrastructures for sensors which can be applied to in-situ and crowdsourced data (Simonis et al., 2016). These standards include Sensor Model Language (SensorML) for describing sensors, Sensor Planning Service (SPS) for the definition of tasks to be performed by sensors and Sensor Observation Service (SOS) for obtaining and

storing sensor observation data. The OGC Web Coverage Service (WCS) and Web Feature Service (WFS) are also used widely to share raster and vector data respectively.

Most scientific applications require multiple resources which cannot be provided by individual GIS software. There is, therefore, a greater need to combine resources from different service providers in a distributed processing manner using web service technology. This is motivated by the evolving concept of web services and service-oriented architectures (SOA). This has further been reinforced by the idea of spatial data infrastructure (SDI) which provides web-based access to data (Schäffer & Foerster, 2008). The OGC Geo-Processing Workflow (GPW) initiative has demonstrated interoperability through chaining of web services in a workflow. Modelling of such workflow can be achieved through the Unified Modelling Language (UML) and Business Process Modelling. However, business process modelling has been widely used for describing workflows. Due to its popularity, this research focused on business process modelling. Modelling visual workflows is facilitated by Business Process Modelling Notation (BPMN) which is a language based on flowcharts for describing business processes (Decker et al., 2010). Another tool commonly used in workflow modelling is the Business Process Execution Language (BPEL) which is an XML-based specification of business processes and their interaction protocols. The graphical object properties supported in BPMN enables the generation of executable BPEL which can be used to implement several geoprocesses that can consume crowdsourced geoinformation, satellite and in-situ data (J. Morales & De By, 2009).

1.2. Problem Statement

Recent technologies such as Web 2.0, web services, lightweight exchange formats as JSON and the ability to process and deliver real-time geospatial data have made it possible to create, share and execute workflows through online browsers which can bring a rich experience to users. Integrating processes and data exposed by RESTful web services can offer great potential for interoperability to enhance shareability and reproducibility of workflows. However, due to the lack of a standardized interchange format for workflows and a platform-independent medium for composing workflows from distributed geoprocessing functions, it is impossible to share and reproduce workflows across different WfMSs. Unfortunately, current WfMSs do not incorporate web services making it difficult to use remote processes. To address these concerns, this research developed a method that can be used to enhance the shareability and reproducibility of workflows. This was accomplished in the following manner.

1. By proposing a standard interchange schema for specifying workflows. This was based on the limitations and strength of existing interchange formats for specifying workflows. Using this standard interchange schema, we establish a method for transforming a workflow from one interchange format to another which is motivated by the concept of OMG model-driven architecture (OMG, 2003).

2. By creating a method for composing workflows from heterogeneously distributed geoprocessing functions using web services. Web services technology driven by service-oriented architecture (SOA) represent a characteristic of platform and language independence which can be explored to achieve interoperability. We implement a web-based prototype system to offer a visual abstraction of the underlying method for workflow composition.

As a proof of concept, we use the prototype system to demonstrate a shareable and reproducible workflow for integration of crowdsourced geoinformation, in-situ and satellite data for water resource monitoring and forecasting. The system allows the user to view and download the result of each step in the workflow composition.

1.3. Research Objective

The main objective of this research is to create a method for enhancing shareability and reproducibility of geoprocessing workflows across different GIS software packages. The method aims to use a standardized workflow interchange format whose JSON schema is derived from the existing interchange formats of different GIS software and established standards. A workflow that combines crowdsourced geoinformation, in-situ measurements, and satellite data is used to demonstrate the applicability of the method as a real-life application using a web-based workflow editor.

There are four sub-objectives to this research;

1. To investigate existing workflow interchange formats and propose an interoperable standard format for sharing workflows.
2. To devise a method for producing shareable and reproducible workflows.
3. To design and implement a prototype that facilitates the creation and sharing of workflows.
4. To demonstrate the applicability of the prototype in combining crowdsourced geoinformation, in-situ measurements and satellite data for water resource monitoring and forecasting.

1.4. Research Questions

Related to the first objective

- i. What are the available tools/software for creating geoprocessing workflows?
- ii. Which interchange formats do they use to share their workflows?
- iii. How can a standard interchange format be created to achieve interoperability?

Related to the second objective

- i. What does it take for a workflow to be shared and reproduced?
- ii. How can a workflow be composed of distributed geospatial web services?
- iii. How can a workflow be shared across different geoprocessing tools/software?

Related to the third objective

- i. How can the prototype system be developed?

- ii. What are the requirements and procedure for setting up the system?
- iii. What are the limitations to this system and the problems that can be encountered?

Related to the fourth objective

- i. What are the potential characteristics of crowdsourced geoinformation, satellite and in-situ data that affects their combination?
- ii. How can specific operations be integrated to combine crowdsourced geoinformation, satellite, and in-situ data?
- iii. What is the added value of the method to shareability and reproducibility of workflow for integration of crowdsourced geoinformation, satellite, and in-situ data?

1.5. Use Case

This research has selected the AfriAlliance project as its use case. The afrialliance project aims to “prepare Africa for future climate change challenges by creating the opportunity for African and European stakeholders to work together in the areas of water innovation, research, policy, and capacity development” (Mannaerts et al., 2017a). As one of its deliverables, AfriAlliance would want to use a multisensory approach to improve water resource monitoring and forecasting in Africa. The triple sensor approach combines different water-related products obtained from satellite, local weather stations, and crowdsourced geoinformation. The following reasons motivated the choice for this use case:

- i. The Triple sensor approach uses three categories of geospatial data that are commonly used in geoprocessing workflows. There are already established standards that define the sharing of these data using web services. Satellite data can be accessed through OGC Web Coverage Service (WCS), in-situ data and Crowdsourced geoinformation using Sensor Web Enablement (SWE).
- ii. Since ITC is involved in this project, access can be provided to the abstract workflow and data for testing.
- iii. This project uses preprocessing and triple collocation methods to combine these datasets. These methods are well-defined and typical functional building blocks to be composed in a workflow using web services by chaining processes from different GIS software processes.
- iv. The shareability and reproducibility of the workflow can be tested by allowing different users involved in the project to rerun the workflow and compare results.

1.6. Thesis Outline

This thesis has adopted the following structure.

Chapter 1 provides a general introduction to this thesis through background information, problem statement and stating research objectives and questions.

Chapter 2 provides a literature review on workflows and how to specify workflows using business process modelling notation (BPMN). The transformation from BPMN to BPEL scripts is also discussed. The chapter looks at a broader perspective of workflows having been first used in business processes then later adopted for scientific processes. The concept of scientific workflows is introduced and terminologies used in workflow including shareability, reproducibility, and provenance are discussed. Finally, this chapter ends by considering factors that affect the reproducibility of scientific workflows.

Chapter 3 discusses software for workflow management, a comparison of how current WfMSs share their workflows and their shortcomings. After that, a discussion on how web-based workflows come to the rescue of current GIS WfMSs and attempts of standardization organizations such as WfMC, Object Management Group (OMG) and OGC to support sharing of workflows through the establishment of standards are addressed.

Chapter 4 discusses the way workflows can be composed by integrating data and processes using web services. The following services are considered: Web Feature Service, Web Coverage Service, Sensor Web Enablement, Web Processing Service and non-OGC compliant RESTful processing services. The OGC Geo-Processing Workflow service chaining is also discussed.

Chapter 5 is based on previous concepts. It introduces a method of producing shareable and reproducible workflows. This chapter also proposes a JSON schema for a standard workflow interchange format. The chapter ends with a discussion of provenance support for reproducibility of scientific workflows.

Chapter 6 discusses the implementation of the proposed system which contains a web-based workflow client and a workflow engine capable of composing a workflow using web services and executing the workflow. The result of the execution is displayed through the web client which also provides users with the ability to download the result using WCS or WFS. Sharing of the workflow is achieved through the standard interchange format which can create reproducible workflows for specific GIS software.

Chapter 7 discusses a proof of concept to demonstrate how the proposed system can be used to solve real scientific problems one of which is the case study. It provides a discussion on the pre-processing of data and the triple collocation method. Creation of shareable and reproducible workflow for integrating crowdsourced geoinformation, satellite, and in-situ data is discussed. To measure the success of the method, the analysis of the result is performed.

Chapter 8 provides a summary of the thesis by answering the research questions and reflecting on the limitations. Moreover, this chapter also suggests a standard workflow interchange format as well as providing recommendations for future work.

2. WORKFLOWS

2.1. Evolution of Workflows

Workflow is a concept that has existed in the business domain for an extended period and has been useful in facilitating the automatic execution of business processes. The last three decades have witnessed the growing trend in the design and use of workflow systems both for business and in scientific research. This has been motivated by the growth of the internet which has opened up the possibility of using workflows to deploy service-oriented applications across wide area networks (Belhajjame et al., 2002). As network capabilities mature and computational power increases, distributed processing powered by web services technology is quickly gaining popularity. This has further been reinforced by the concept of spatial data infrastructure (SDI) which provides web-based access to data (Schäffer & Foerster, 2008).

As the use of workflows increases, the need for a universal standard to facilitate the creation, sharing, and reuse of workflows becomes a necessity. In 1993, the Workflow Management Coalition (WfMC) was created to promote and develop the use of workflows through the establishment of standards for software terminology, interoperability and connectivity among business processes (Schmidt, 1999). They developed a large set of reference models, documents and standards with the main focus on business processes. Three years later, they came up with a formal definition for workflows as *“the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules”* (Barga & Gannon, 2007). Business processes relate to a great extent to this definition since they involve the shift in tasks from one person to another. However, the current evolution of workflows is based on service-oriented architectures in which the tasks are carried out in a distributed environment using remote computational resources (Curcin & Ghanem, 2008). This contrast is the difference between business workflows and scientific workflows. The workflow logic of business processes is control flow driven making their execution robust which is a contrast to their counterpart scientific workflows which are data flow driven. Scientific workflows often utilize a lot of computing and storage resources which cannot be adequately provided by a single computer. As a result, most of the processes in scientific workflows are executed remotely and are coordinated by a workflow management system. The business workflows are always not fully automated as compared to scientific workflows (Sonntag, Karastoyanova, & Deelman, 2010). They involve the use of humans in some stages of the execution process whereas, in scientific workflows, the humans are only required during the creation of the workflow. Despite their differences, scientific workflows borrow a lot from the original concepts of workflows which were based on business processes.

2.2. Workflow Modelling

Modelling workflow can be achieved through the Unified Modelling Language (UML) and Business Process Modelling. However, business process modelling has been widely used for describing scientific workflows. Business Process modelling is an essential component to the success of software development. Morales & De By (2009) observed that the business process modelling field strongly drives workflow modelling. Business process modelling uses one of the most popular conceptual modelling tools for specifying workflows known as the business process and model notation (BPMN). BPMN offers a graphical notation for high-level modelling using descriptive and analytic constructs. Business process and model notations were developed as a result of an agreement among several tool vendors towards a standard of notations for describing business processes (Burattin, 2015). Since the release of the first flowchart-based BPMN in May 2004, BPMN has gained a wide audience both in business processes and scientific processes. BPMN has enabled users to create sequences of processes and their supporting information in a graphical representation which describes a business process. Figure 2.1 shows essential OMG (2011) BPMN elements which include activities, events, gateways, connectors.

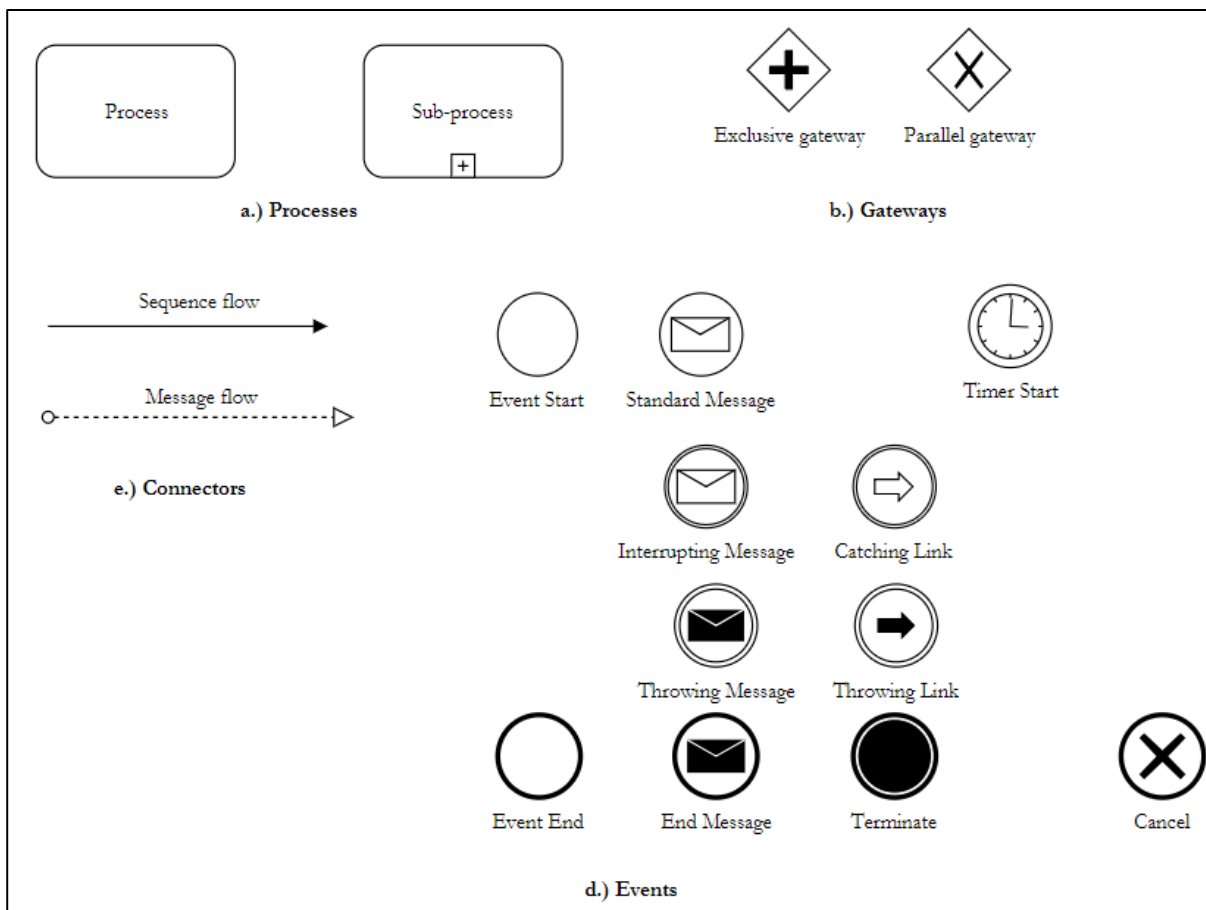


Figure 2.1: Basic BPMN elements

Activity: Activity identifies the task performed by a company. They are often represented as rectangles with rounded corners. BPMN specifies several types of tasks depending on their roles which include

service tasks, send task, receive task, user task, manual task, business rule task, script task. For this research, we use service tasks since it is used for web services or automated applications.

Gateways: Gateways are used to control the flow of processes using sequence flows through divergence and convergence in a process.

Sub-activity: sub-activity can be used to hide the different level of abstraction of a task.

Events: The user or the system always trigger events. An event can be used to start the execution of a process, pause or terminate it.

Connectors: These components are used to indicate the flow of information or association.

Once a business process has been specified using the business process modelling notation, it is saved as a BPMN document. A BPMN document is an XML based file representation of the graphical workflow. BPMN documents by their own cannot be executed. Therefore, there is a need to convert them to an executable specification which is written in the business process execution language (BPEL). BPEL can be thought of as an XML-programming language for web services compositions since it is used together with Web Service Definition Language (WSDL). Moreover, BPEL incorporates several features of web service development including XML data definition and manipulation, a dynamic binding mechanism which is based on the explicit manipulation of endpoint references and declarative mechanism for correlating messages to process instances, an essential requirement for asynchronous communication (Ko et al., 2009). BPMN's graphical standards are graph-oriented representing logical flow through nodes and connectors whereas BPEL execution standards are block oriented in which the flow of execution is controlled by nesting different kinds of syntactic control primitives using XML.

A visual workflow obtained from a BPMN document can be serialized as a BPEL script before it can be executed. To do this, the non-linear workflow has to be transformed into a linear workflow to establish the sequence of its execution. A Scientific workflow forms a directed acyclic graph (DAG) in which the nodes represent the participating services whereas the edges represent data flow between services (Schäffer & Foerster, 2008). A DAG exhibits three properties which include reflexivity, asymmetry, and transitivity.

Given a set $O = \{A, B, C, D, E, F, G\}$ to represent the elements of a scientific workflow,

- Reflexivity of A is defined by $A \leq A$
- Asymmetry: if $A \leq B \rightarrow B < A$ is false
- Transitivity: if $A \leq B$ and $B \leq C$ then $A \leq C$

Once all the elements of the set have been modeled as DAG, their topological relationships can be determined to create a linear ordering of the processes as shown in Figure 2.2. Several permutations of the ordered processes for the DAG can be obtained for similar illustrations. These include A-B-C-D-E-F-G or A-B-C-D-E-G-F or A-C-B-D-E-F-G or A-C-B-D-E-G-F. For a given DAG $G(K, E)$, the topological sort of its vertices is a sequence $S = \{u_1, u_2, \dots, u_n\}, u \in K$ in where every element of K appears

exactly once (Schäffer & Foerster, 2008). With this notation, no process in a workflow can be repeated in the sequence. The concept of the DAG is used in Section 6.5.2 to determine the execution order of processes in a workflow.

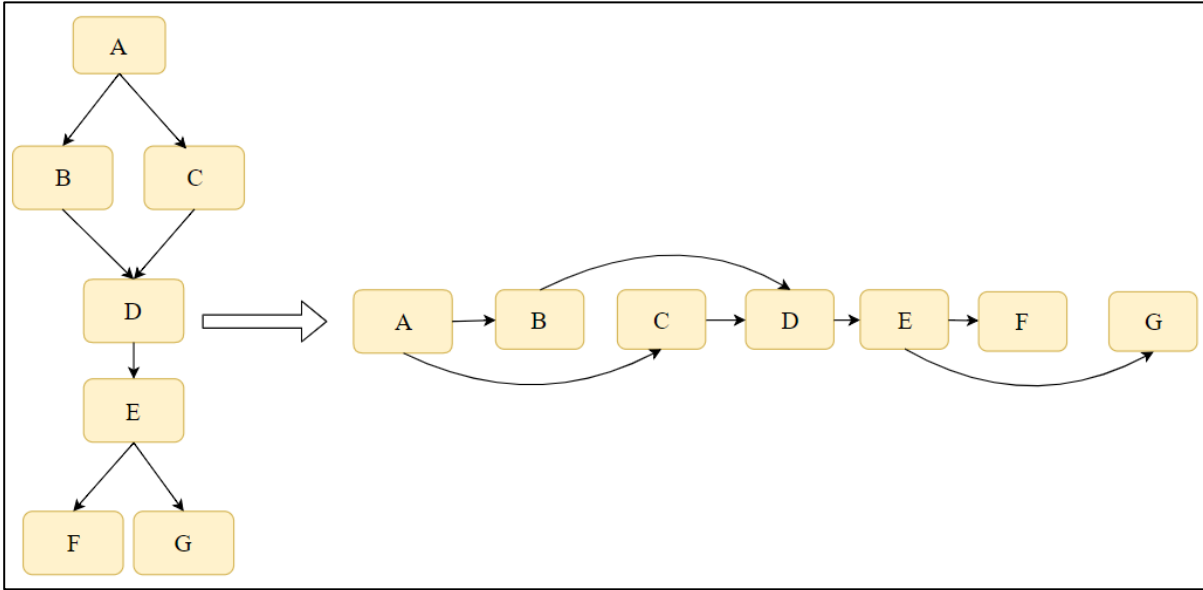


Figure 2.2: Topological Sorting of Processes using DAG.

Adopted from (Schäffer & Foerster, 2008)

2.3. Scientific Workflows

Application of workflows to scientific calculations, simulations, and experiments were much inspired by the success witnessed in the application of the workflow management system to business processes (Sonntag et al., 2010). Researchers in various domains have embraced the use of scientific workflows to conduct a range of analysis and scientific pipelines since they model computation structure and data processing tasks in a manner that help in the management of a scientific process. Lemmens et al. (2018) distinguish scientific workflows into two levels of abstraction, abstract and concrete workflows. Abstract workflows are used to provide an overview of the operations, their input, and output without having to specify data sources and operation parameters. An abstract workflow hides the implementation details of a workflow and can be considered as platform independent.

On the other hand, a concrete workflow provides details of steps of processes of a workflow which can be executed by a particular WfMSs. Given an abstract workflow, it is possible to generate its concrete workflow which can be implemented in different software. Scientific studies have proved that it is possible to automatically generate concrete or executable workflows from abstract workflows using their semantic descriptions. For instance, Ubels (2018) researched on automatic conversion of abstract workflows to executable using semantic web technologies. This research opened the way for scientific processes and workflows to be discovered using ontologies and semantic web technology thus supporting shareability of scientific processes. Scheider & Ballatore (2018) also proposed a method for expressing workflows as linked data which is easily publishable and discoverable through the web. Their method provides support

for searching, interpreting and reusing workflows in a modular manner using semantic descriptions. In the following subsections, we discuss some of the concepts of scientific workflows.

2.3.1. Provenance

The term provenance means source or origin. It can be applied to data to indicate its evolution and modification applied to it (Juhnke et al., 2010). To ensure reproducibility and repeatability, sufficient provenance information is desired. Just as with data, provenance can be used with workflows to capture information such as processes and their execution environment, input parameters provided to processes, a log of processes, connections, intermediary and final outputs. There are seven different scenarios explored by Taylor et al. (2007) where workflow provenance information can be relevant. Some of these scenarios include to repeat a workflow execution, to reproduce a data output by retrieving intermediate results or inputs from which these outputs were derived, to assess the performance of a service that has been invoked multiple times and to debug a failed workflow execution in order to establish which service failed and the possible causes.

A. Banati et al. (2015), identified four levels of provenance, i.e., system, environment, data, and workflow model. The system level provenance helps answer the questions of what, where, when and how long has been executed by storing the type of infrastructure, the variables, and the timing parameters. System-level provenance ensures portability of the workflow. The environmental provenance stores the execution details including the operating system properties, libraries, and code interpreter properties. Data provenance deals with data lineage and additional provenance information like input(s) and output(s) names, types, size, parameters significance, among others. The workflow model provides lineage information of the workflow which documents the history of its modification. The provenance information collected at the fourth level is necessary for workflow versioning.

2.3.2. Shareability

Shareability of scientific workflow is defined as the ability to transfer the workflow from one scientist to another or one environment to another in a manner that allows readability and understanding of the workflow that is not necessarily created by the same scientist or in the same environment. Sharing workflows helps scientist to understand scientific processes created by their colleagues as well as make the workflows as an essential building block in their new processes. Most GIS workflow management system enables the creation of workflow but sharing of these workflows across the different system is still not possible. This affects interoperability between GIS workflow management system forcing scientists to recreate their workflows in different environments. An attempt to achieve interoperability between scientific workflow management systems was undertaken by A. Banati et al. (2015) who developed Gefyra, which is a system based on the PROV workflow model to translate provenance information from one format to another. However, PROV was not entirely successful since every scientific workflow management system could not use it.

2.3.3. Reproducibility

Reproducibility is the most vital part of science enabling scientists to evaluate the validity of each other's methods and hypothesis by running an experiment at different locations using different tools (Gil et al., 2007). Reproducibility allows a workflow specified to address a particular scientific problem to be reused by different users under equivalent conditions without having to manipulate or change the original specification to produce scientifically similar results. To reproduce scientific workflows, provenance information must be collected on the individual tasks, their execution environments as well as their input and output parameter requirements. Rich provenance information, as well as careful workflow design and documentation, are necessary for efficient workflow reproducibility (Anna Banati et al., 2016).

Shareability and reproducibility of workflows are important application requirements towards achieving interoperability and accessibility of geospatial resources which includes data and processes. Shareability needs a mechanism in which processes and data can be exchanged between different WfMS by use of a standard interchange format. Semantically enabled exchange formats; for instance, JSON provides an interoperable way in which humans and machines can share workflows. Shareability is mainly concerned with preserving the physical representation of the workflow and data flow between processes whereas reproducibility is responsible for the logical preservation of the workflow by which rich provenance information is used. Reproducibility makes it possible to reuse workflows created by others to verify the correctness of their intermediate results or hypothesis (Bechhofer et al., 2013). A reproducible result or method of a scientific experiment would require the use of similar processes, data and conditions in a workflow.

Shareability and reproducibility are strongly related concepts. Workflows cannot be reproduced if they are not shareable across different environments. A workflow should always be reproducible; otherwise, it has no value. However, the most important considerations are the threshold and conditions under which it can be reproduced.

2.4. Factors Affecting the Reproducibility of Workflows

Zhao et al. (2012) came up with the term *workflow decay* to refer to a situation where a workflow is not able to be reproduced. In other literature materials, this situation is known as *workflow irreproducibility*. They revealed that nearly 80% of workflows could not be reproduced due to volatile third-party resources, missing data, missing the execution environment and insufficient metadata about the workflow. We discuss each of these factors in the subsequent sections. Figure 2.3 provides an overview of their findings on how the four factors affect reproducibility. It can be observed that third-party resources greatly affect reproducibility of workflows whereas execution environment has the least influence in reproducibility of workflows.

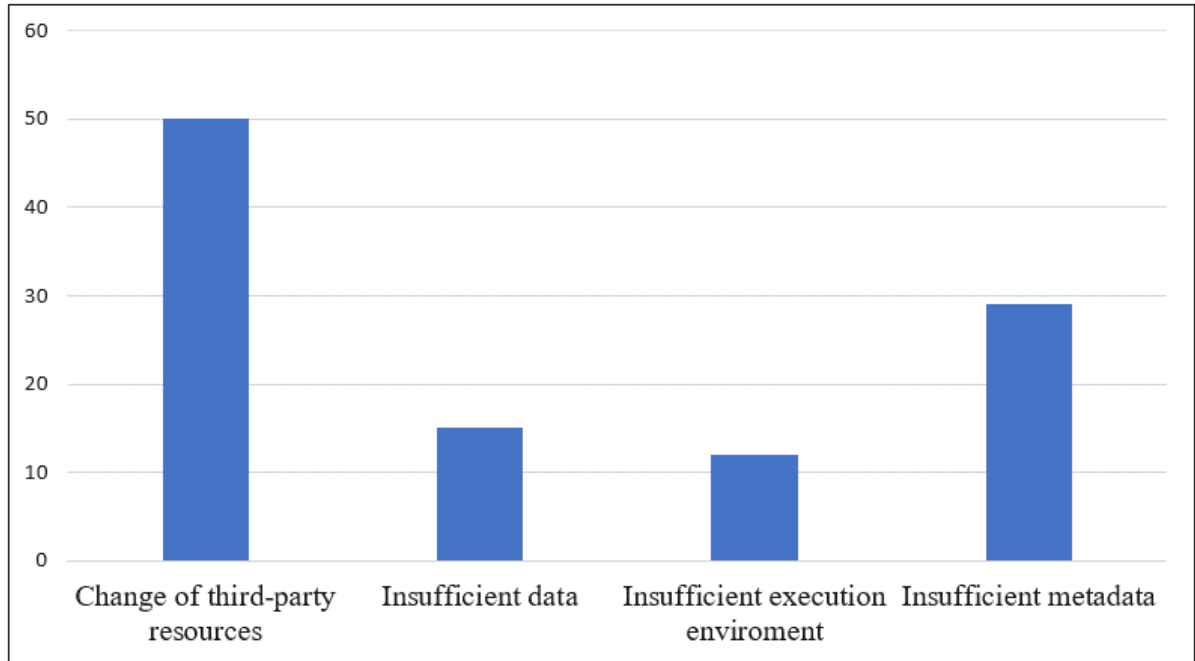


Figure 2.3: Comparison of the causes of workflow decay.

Source: (Zhao et al., 2012)

2.4.1. Third-party resources

Third-party resources include web services and databases which are used in the instantiation of a workflow. Provision of such services may be changed or interrupted thereby interfering with the execution of the workflow. The provider of a web service may change the configuration and implementation of the web service thereby giving a different result or making it impossible to execute a workflow. Figure 2.4 illustrates that unavailability of third-party resources contributes greatly to the irreproducibility of workflows. This is due to the depreciation of web services and server failures which are not consistently administered. Inaccessibility of resources includes the use of different identifiers from the one previously used in composing the workflow, introduction of access rights requiring authentication to use a service. Updates to web services result in changes on the types and quality of the outputs due to software or library upgrades and also changes in functionality as a result of making references to a different web service using the same identifier.

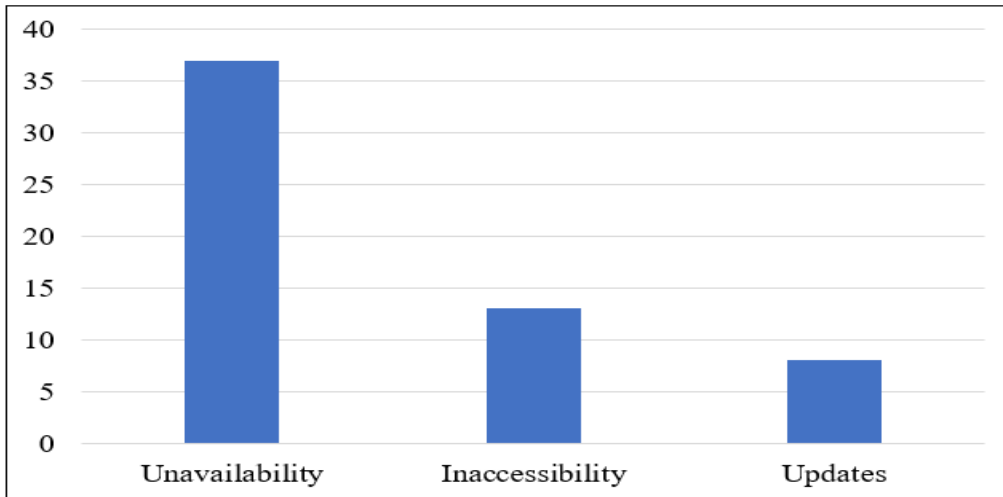


Figure 2.4: Comparison of Workflow decay due to third-party resources. Source: (Zhao et al., 2012)

2.4.2. Nature of the input data

Insufficient input data also affect the reproducibility of scientific workflows. Whenever a mandatory data required by a process cannot be found, the execution of the entire workflow fails. In the case of geoscientific workflows, data varies by scale, resolution and coordinate system. Data of different scale, resolution and coordinate system are incompatible. When incompatible data are used together, they introduce errors which affect the execution of the workflow.

2.4.3. Execution environment

Execution environment of a workflow comprises of the software and libraries that are required to execute the workflow. Whenever there is a missing library from which a process depends, reproducibility of the workflow is affected. Software and libraries are also prone to regular updates from their vendors. Such updates can introduce compatibility with the old implementation of the workflow.

2.4.4. Workflow Metadata

Workflow metadata is part of the provenance information that is required to reproduce a workflow. Insufficient description of the workflow negatively affects the reproducibility of scientific workflows. These include the description of processes, input/output data, the flow of information (connections), purpose and expected outcomes of the workflow. When there is no adequate information which describes the workflow, users are not able to understand its purpose and the expected result associated with it. From their study, Zhao et al. (2012) established that 28% of workflow irreproducibility is caused by insufficient descriptive information about the workflow.

3. WORKFLOW MANAGEMENT SYSTEMS

The Workflow Management Coalition (WfMC) defines a Workflow Management System (WfMS) as “a system that completely defines, manages and executes workflows through the execution of software whose order of execution is driven by a computer representation of the workflow logic.” A WfMS is made up of two main components, the workflow client and the workflow engine. The workflow client contains a graphical editor which allows users to interactively compose the visual workflow by dragging and dropping the figures representing the workflow elements. It also allows users to define the rules and sequence of execution of the workflow.

Additionally, the workflow client has a monitor where users can view the result of their workflow once execution is completed. After visually modelling a workflow, it is translated to a script which is sent to the workflow engine where execution takes place. The order of the execution is based on the sequence defined by the user when composing the workflow. Upon reaching the workflow engine, the workflow has to be transformed into a linear workflow to determine its execution order. How the workflow engine arrives at this linear ordering has been mentioned in Section 2.2. A similar concept has also been discussed by (Schäffer & Foerster, 2008). Figure 3.1 below visually illustrate the composition of the workflow management system.

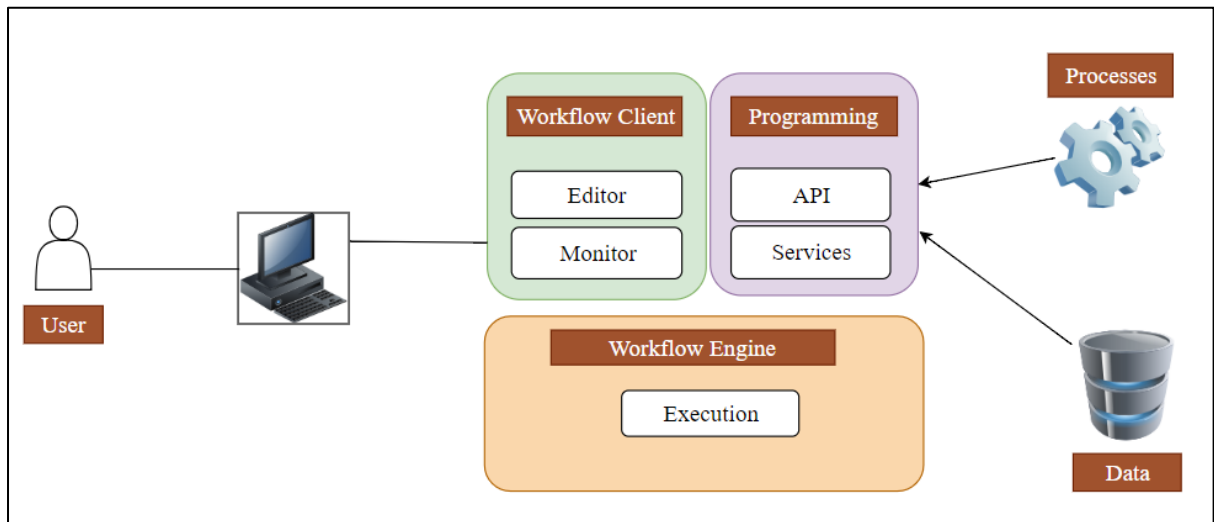


Figure 3.1: Composition of Workflow Management System

Standard organizations like OMG, WfMC, and OGC have established several standards to support the development of workflow management systems. These standards enable WfMSs to automate and coordinate tasks by independently developed applications distributed by different software vendors (Schmidt, 1999). Some of the popular standards established by the standardization organizations are discussed in Section 3.1. As workflow technology get more appreciated within the scientific domain as a way to automate processing, new WfMSs are increasingly evolving. However, most of the emerging proprietary WfMSs define their standards for their workflow management. This trend has made it possible

for this research to group WfMSs into two broad categories. These include standardization compliant WfMSs and non-standardization compliant WfMSs. The following Sections discuss these categories as well example of popular WfMSs in each category.

3.1. Workflow Specification Standards

One of the popular standards used in specifying workflows is OMG's Business Process and Management Notation (BPMN). We introduced BPMN in Section 2.2 and discussed how it uses notations to visually model workflows utilizing an industry standard exchange format. These notations are generally accepted and understood by standardization compliant WfMSs thereby ensuring shareability and reproducibility of workflows. The OGC has also specified several standards that have made the composition of workflows through web services possible. These standards discussed in Chapter 4 include WPS, WFS, WCS, and SWE. WPS provides a specification for enabling sharing and accessing of processing functions while the other standards specify how a satellite and sensor data can be shared. The OGC process chaining also defines three ways in which a workflow can be created by chaining several web services. Section 4.4 provides a detailed description of the OGC process chaining.

3.1.1. BPMN

Business Process Model And Notation Specification Version 2.0 provides a list of machine consumable documents which describes the schema of a business process. The most relevant for this research are the five XSD files which define the process semantics and its graphical representations. These XSD files include BPMN20.xsd, Semantics.xsd, BPMNDI.xsd, DC.xsd and DC.xsd. BPMN documents have several elements and attributes whose descriptions are well elaborated in the BPMN 2.0 specification by (OMG, 2011). This research focuses on some of these elements which are relevant to our proposal of a standard schema for workflow interchange. At the top-level schema, a BPMN document contains the elements; *process* which specifies the semantics of the workflow and the *BPMNDiagram* for the graphical representation. The attributes of the *process* are *id*, *name*, and *isExecutable*.

- *id*: Represents the identifier for the process
- *name*: Represent the name of the process
- *isExecutable*: This is a Boolean value specifying whether the business process is executable or not.

The *process* has sub-elements such as the *startEvent*, *endEvent*, *sequenceFlow* and *serviceTask*. The *startEvent* indicates the beginning of the process while the *endEvent* indicates the end of the process. The *sequenceFlow* outlines the flow of activity from one task to another. It resembles the connections between activities. The *serviceTask* is a type of BPMN task specifically meant for web services or an automated application. It references an operation and includes attributes such as *id*, *name*, and *implementation*. The *implementation* attribute specifies the web service technology or URI that is used to send and receive messages. BPMN captures data requirements as *dataInput* and *inputSet* while the result of execution is captured using

dataOutput and *outputSet*. The *ioSpecification* or *InputOutputSpecification* is the parent class from which the data inputs and outputs and input sets and output sets are derived.

Table 3.1: BPMN Process Elements

<i>Elements</i>	<i>Attributes</i>	<i>Sub elements</i>
ioSpecification	Id	dataInput dataOutput inputSet outputSet
startEvent	id, name	
endEvent	id, name	
sequenceFlow	id, name, sourceRef, targetRef	
serviceTask	id, name, implementation	Incoming Outgoing ioSpecification dataInputAssociation dataOutputAssociation

Table 3.2: BPMN Diagram Elements

<i>Elements</i>	<i>Attributes</i>	<i>Sub elements</i>
BPMNPlane	id, bpmnElement	BPMNShape BPMNEdge

BPMNShape depicts a BPMN model element and contains a screen coordinate for the visual representation of the element which can be an event or activity. *BPMNEdge* is used to depict the relationship between BPMN model elements.

3.1.2. OGC Geoprocessing Workflow (GPW)

The OGC Geo Processing Workflow (Werling, 2008) wraps several web processing service (WPS) in a BPEL script. This makes it possible for creating workflows through web services and executing them in a BPEL engine. However, BPEL alone cannot visually represent a workflow and therefore relies on BPMN. The OGC process chaining also specifies how a workflow can be created by chaining several services in a single WPS execute request. However, this approach cannot be useful when the services are offered on different servers. Since the OGC GPW has not been fully established as a standard, this research focused on the WPS process chaining. The attributes and elements of a WPS which were considered relevant for this research were as shown below.

Table 3.3: OGC WPS Process Elements

Element	Attributes	Sub Elements
Identifier		
Title		
Abstract		
DataInputs		Input
ResponseForm		RawDataOutput
Input	maxOccurs minOccurs	Identifier Title Abstract ComplexData/ LiteralData Data
ComplexData		Format Reference
LiteralData		Data
Reference	mimeType xlink:href method	
RawDataOutput	mimeType	Identifier

The *Identifier* elements refer to the unique identity of the WPS process. *DataInputs* specifies the input data requirements. An operation can have at least one input parameter. The *minOccurs* and *maxOccurs* attributes specify the number of input parameter requirement. A mandatory input has a *minOccurs* value of 1 and above while an optional input has a *minOccurs* value of 0. An input has other elements like *identifier*, *title*, *abstract* (description) and the value which can either be passed as a reference or by value. *ComplexData* is used for spatial data types like coverages and vector data whereas *LiteralData* is used for non-spatial data types like string, numeric and Boolean.

3.2. Standardization compliant WfMSs

The BPMN website¹ provides a list of several WfMSs that implements the BPMN 2.0 specification. For this study, we identified a few of these WfMSs that have been used by experts in the geospatial domain for managing processes in a workflow. Some of the WfMSs that were found to have demonstrated the

¹ <http://www.bpmn.org/>

applicability of these standards were Camunda², JBPM³, Bonita⁴, ProcessMaker⁵ and Yaoqiang BPMN editor⁶. The OGC Testbed-13 Pross & Christoph (2018) demonstrated the combination of BPMN and OGC WPS for a conflation workflow using the Camunda BPMN Engine. Rosser et al., (2018) also took advantage of the JBPM engine to demonstrate metadata profiling approaches for geoprocessing workflows. Since these tools allow sharing of workflows as BPMN documents, it is possible to reproduce the same workflows among other BPMN compliant software. ProcessMaker is a cloud-based BPMN compliant software that is multi-tenant and scalable for multiple users without any management overhead or performance issues. ProcessMaker allows users to connect to remote databases and retrieve data which can be used as inputs to the workflow. Additionally, it offers the ability to integrate third-party functions and libraries as well as connect a set of web services using their REST API.

3.3. Non-Standardization Compliant WfMSs

There exist many WfMSs that do not conform to any established standard for specifying workflows. Among these are the popular WfMSs used within the domain of geographic information science (GIS). This research focuses on four main software packages that are frequently used by scientist in the GIS domain. These include ILWIS model builder, ArcMap model builder, QGIS processing modeler, and ERDAS Imagine Spatial Modeler. These tools provide users with the ability to create visual workflows following the standards of business process modelling notations (BPMN). Visual workflows provide an abstract view of the underlying system definition of the process thereby making it simpler for people with little knowledge to understand the workflows. In as much as these tools borrow the BPMN diagram notations, they use their file formats and structure to represent their workflows.

Most the popular software packages support sharing of workflows using a semantic web-based exchange format which can be understood by both machines and humans. However, these formats are not based on any standardized schema like the BPMN-based WfMSs which we discussed in the previous Section. This makes it difficult to achieve interoperability among different WfMSs. XML and JSON are the two commonly used web-based exchange formats today. The latest version of ILWIS uses a semantic web-based exchange format in JSON-LD which makes sharing of workflows possible (Lemmens, Schouwenburg, et al., 2018). ERDAS allow users to share their workflows in JSON file formats while QGIS support both XML and JSON file formats. ArcMap model builder, however, does not support sharing of workflows using any of the exchange formats discussed above. Therefore, we did not give it much attention in this research. Apart from the GIS WfMSs, there are also other commonly used WfMSs such as KNIME and Taverna which uses their specifications for their workflows. KNIME provides users

² <https://camunda.com/>

³ <https://www.jbpm.org/>

⁴ <https://www.bonitasoft.com/>

⁵ <https://www.processmaker.com/>

⁶ <https://sourceforge.net/projects/bpmn/>

with an editor for visually creating workflows and an engine for executing already created workflows. It is an Eclipse-based tool which is available as a desktop application and commonly runs on Java environment.

3.3.1. ILWIS Model Builder

The structure of an ILWIS workflow reveals four main elements which are *id*, *metadata*, *operations*, and *connections*. The metadata is a JSON object of attributes which describe the workflow. The operations contain a list of processes that are used in the workflow while the connections list the sequence of connections between operations and the parameters. Each operation has an id, metadata and an array of inputs and outputs. The attribute *inputparametercount* indicate the number of required input parameters for operation while *outputparametercount* indicates the number of required output parameters. The resource specifies the execution engine of the operation. By default, this is always assigned the value “ILWIS.” The *syntax* attribute specifies the internal name of the operation as a function of inputs.

Table 3.4: ILWIS Workflow Elements

Element	Attributes	Sub-element
workflows		
id		
metadata	description inputparametercount outputparametercount longname resource syntax	
operations		id Inputs Metadata outputs
Operation Metadata	description inputparametercount outputparametercount keywords label longname resource syntax final	

Input	id change local description name optional show type URL value	
Output	id local description name optional show type URL value	
connections	fromOperationID toOperationID fromParameterID toParameterID	

3.3.2. QGIS Processing Modeller

QGIS modeler provides a JSON file export option for its workflow but with very different structure or format. At the top level of the JSON file are two elements *values* and *class*. *Values* represent the semantics of the workflow whereas *class* has a default value of

“processing.modeler.ModelerAlgorithm.ModelerAlgorithm” which indicates that the file is a modeler specific type. The most relevant elements of *values* are *inputs* and *algs*. The *inputs* element is used to specify modeler spatial input parameters. The attributes of an input parameter include the inputs id, screen (x, y) position, value, optional, description, data type, name, and others which are irrelevant for this research. The *algs* define the algorithms or operations used in the workflow. A particular algorithm is assigned a key with a set of elements which include input parameters, output values, the name of the algorithm, internal name of the algorithm (*consoleName*), description and the screen coordinate.

Table 3.5: QGIS Workflow Elements

Element	Attribute	Sub-element
class		
values		Inputs helpContent group name algs
Inputs	Id (key derived from the input name and index) pos (x, y) name value optional default description data type	
algs	Id (key derived from the algorithm name and index) Name consoleName description pos (x, y)	Params (input parameters) outputs
params	Name (derived from the input id)	
outputs	Description pos (x, y)	

3.4. Shortcomings of Current WfMSs

Observation of the WfMSs discussed in the previous Sections revealed a lot of differences in the manner in which they specify and share their workflows. These differences can affect the execution and visual representation of a workflow outside its proprietary software package. To help us understand these limitations better, this research adopted the following questions.

i. Which exchange format is used?

This question is intended to provide answers to the formats used to exchange workflows by specific WfMSs. The formats can include XML, JSON, text files, script file such as Python or batch, etc.

ii. Does the schema of this format conform to any standard?

This question addresses the grammar used in specifying the workflow and determine whether it is based on a standard. Standards for workflow specification include BPMN, OGC GPW among others.

iii. Is the workflow reproducible from this format and schema?

This question determines if it is possible to reproduce the workflow based on the answers from the previous questions.

iv. Does it store enough metadata to describe a process?

From the discussion in Sections 2.3 and 2.4, we found out that sufficient provenance information is required to support reproducibility of the workflow. In Section 5.1 we provide a minimum requirement for metadata information that is sufficient to describe a process. Therefore, this question is intended to answer if the selected WfMSs adhere to such a requirement.

v. Does it support workflow composition from remote services?

The discussion on remote services required to compose a workflow is discussed in Chapter 4. By this question, we intend to determine if the selected WfMSs are capable of composing and executing workflows from web services, e.g. WPS, WFS, REST services, etc.

The findings from the questions above are shown in Table 3.6. These findings are further discussed in the following sections.

Table 3.6: Observed Differences among selected WfMSs

Property	ILWIS	QGIS	ERDAS	ArcGIS	BPM tools	OGC GPW	KNIME
Which exchange format is used?	JSON	JSON & XML	JSON	Python	XML	XML	XML
Does the schema of the format conform to a standard?	No	No	No	No	Yes	Yes	No
Is the workflow reproducible from this format?	No	Yes	Yes	No	Yes	Yes	Yes
Does it store enough metadata to describe a process?	Yes	No	Yes	No	-	Yes	-
Does it support workflow composition from remote services?	No	No	No	No	Yes	Yes	No

3.4.1. Standardization compliant WfMSs

Even though some of these WfMSs have been proved by researchers to be suitable for composing and executing geoprocessing workflows, this research, however, observed the following limitations associated with them.

1. These WfMSs, for instance, Camunda modeler and Yaoqiang BPMN editor only provide client specific functionalities like editing of workflows. They cannot be used to execute workflows. To execute the workflow, the user is required to use a different workflow engine. This makes it difficult to automate the execution of workflows.
2. Since these WfMSs are based on BPMN's XML schema, they do not support other exchange formats like JSON which has been proved to be lightweight and suitable for sharing workflows through the web (Nurseitov, Paulson, Reynolds, & Izurieta, 2009).
3. They specifically target business processes and require expert knowledge to use for geoprocessing workflows.
4. Some of the WfMSs are commercial and thereby require users to pay to get full functionalities.

3.4.2. Non-Standardization Compliant WfMSs

This category of WfMSs are popularly used within the geo domain and offer great benefits to users when it comes to automation of geoprocessing tasks. However, observation of these systems revealed several limitations which make it difficult to share and reproduce geoprocessing workflows.

1. Unlike the BPMN-based WfMSs, this category of WfMSs does not have a standard schema for sharing their workflows. They have their file formats and use a different structure to define their workflows.
2. It is not possible to recreate the visual workflows from the file formats of the workflow produced by some of the WfMSs. For instance, ILWIS does not store the (X, Y) coordinates of the visual components in its JSON structure. Even though this doesn't in any way affect the execution of the workflow, it could pose a more significant challenge to recreate a visual representation of the same workflow in a different environment.
3. Inadequate metadata information attached to the workflow. For instance, QGIS store data types for input parameters but not for output parameters. It is thus challenging to assign the output of one operation to a different operation which might be using different data types. It requires one to have prior information about the expected output parameter data type. However, in the absence of this information, it is not possible to connect from one process to another. Non-spatial data (texts, Boolean, numeric) have no proper definition as other inputs of spatial types in QGIS. For example, in case of a vector data inputs, QGIS store value, data type, optional (true/false), name and description attributes whereas for numeric data inputs they only store value and name attributes.
4. Some of the workflow interchange file formats are not reproducible, e.g., ILWIS cannot reproduce a workflow of its JSON format, and ArcMap cannot do the same for its Python file formats.
5. They do not allow composition of workflows from web services, therefore, cannot support distributed computing.

This research summarizes the limitations of the two categories of WfMSs using the following way.

- The workflow definitions created by different WfMSs are not interoperable because they are not based on a universal standard. Therefore, other workflow engines cannot read and execute workflows produced by different WfMSs. Even though interoperability has been demonstrated with the sharing of data from one WfMS to another, the same is not possible for processes and workflows.
- There is no standard workflow interchange schema to map from one workflow to another. For instance, it is not possible to create a BPMN document from ILWIS or QGIS workflow.
- There is little metadata information attached to the workflows making it difficult to reproduce the same methods in different WfMSs. Minimum required metadata information is discussed in Section 5.1.
- Some of the workflow interchange formats are not reproducible, e.g., ILWIS (JSON) and ArcMap (Python) file formats.
- Mapping on endpoints of third-party resources is not possible making it difficult to discover and use processes owned by different service providers.

The WfMSs does not allow composition of workflows from web services. Furthermore, the GIS software does not expose their operations as web services even though OGC proposed the WPS standard in 2007 which can be used to expose GIS operations.

3.5. Proposed Solution for the Challenges facing Current WfMSs

To achieve interoperability, WfMSs should be able to create shareable and reproducible workflows. In the previous section, we discussed the current WfMSs and the challenges they face which can affect sharing and reproducibility of workflows. This research considers two approaches which can be used to eliminate the challenges with current WfMSs.

The first approach focuses on the establishment of a standard workflow interchange format which can be adopted by developers of geoprocessing software packages. This has been motivated by the differences observed in the schema for the workflow interchange formats of the different WfMSs. Standardized interchange format is needed to import a workflow created in a different environment. The standard schema of BPMN, for instance, supports the translation of a graphical BPMN document to execution standards of BPEL and also to exchange scientific workflows between different software packages (Mendling, Mendling, & Neumann, 2004).

Apart from the need to have a standard interchange format for sharing workflows, the current WfMSs pay more attention to modelling simple and static process thus does not offer sufficient flexibility for heterogeneous distributed processing using web services. This makes it impossible to achieve high-level interoperability and integrate workflow processes from different GIS systems and service providers (A. Banati et al., 2015). Web services and ontology technology is driven by service-oriented architecture (SOA)

and represent a characteristic of platform and language independence which can be explored by current WfMSs to achieve interoperability. The two approaches are discussed further in the following subsections.

3.5.1. Standard Workflow Interchange Format

The specification and standardization of workflow interchange format are required to achieve interoperability among different scientific applications (Mendling et al., 2004). An interchange format describes the structure of a file through grammar or schema for a particular application domain. A standardized interchange format supports the integration of applications by allowing independent software components to consume data files produced by other software packages. International bodies have been able to come up with several standards for creating and describing geospatial processes. WfMC, for instance, came up with XML process definition language (XPDL) in 1998 as an interchange format for business process models. Its popularity was further enhanced when WfMC endorsed BPMN as a graphical standard for business processes in 2004 (Ko et al., 2009). Object Management Group also introduced business process definition metamodel (BPDM) in 2004 as a rival interchange format to XPDL. Its interchange format is defined by an XML schema and XML for Metadata Interchange (Amsden et al., 2004). However, it was outshined by the XPDL due to the long history of XPDL, stability and strong industry support from WfMC. BPDM has received a lot of criticism as complex and user-unfriendly standard.

A. Comparison of XML and JSON exchange formats

The XML based formats have existed for decades and were adopted by international bodies to create standards for data exchange. For instance, WfMC and OGC standards are mostly XML based. However, the advancement in technology has presented another file format which is easier for computers to parse. JavaScript Object Notation popularly referred to as JSON provides an alternative to XML based file formats because it parses up to one hundred (100) times faster than XML (Nurseitov et al., 2009). Representation State Transfer (REST) architecture has quickly replaced the traditional Simple Object Access Protocol (SOAP) architecture in the past few years because of its ease of implementation and use. JSON is mainly used with REST architecture because of its lightweight. In as much as most of the standards still use XML and SOAP, observation of the current trends in web service technology shows a decline in their use in favor of RESTful services and JSON as an exchange format. Using an outdated technology to specify standards has the potential to lower the applicability of a standard. Due to this, this research adopts JSON as the data format for the standard workflow interchange.

B. Workflow Transformation

Having a standardized workflow interchange format, we can perform a transformation of workflows from one WfMS to another using mapping rules specified by the constructs in different WfMSs. The concept of

our workflow transformation is motivated by the OMG Model Driven Architecture (MDA) which tries to separate application concerns the underlying implementation technology. J. M. Morales (2004) defines a model transformation as a set of rules that describe how a model in the source language can be transformed into a model in a target language. In this definition, a transformation rule specifies how a construct in one language can be mapped to a construct in a different language. This research adopts this definition for workflow transformation to mean a set of rules that describe how a workflow in the source WfMS can be transformed into a workflow in the target WfMS. MDA established three types of models which are Computation-Independent models, Platform-Independent Models and Platform Specific Model (OMG, 2003). Each of these models is implemented at different layers in the architecture and offers an abstraction of the underlying constructs.

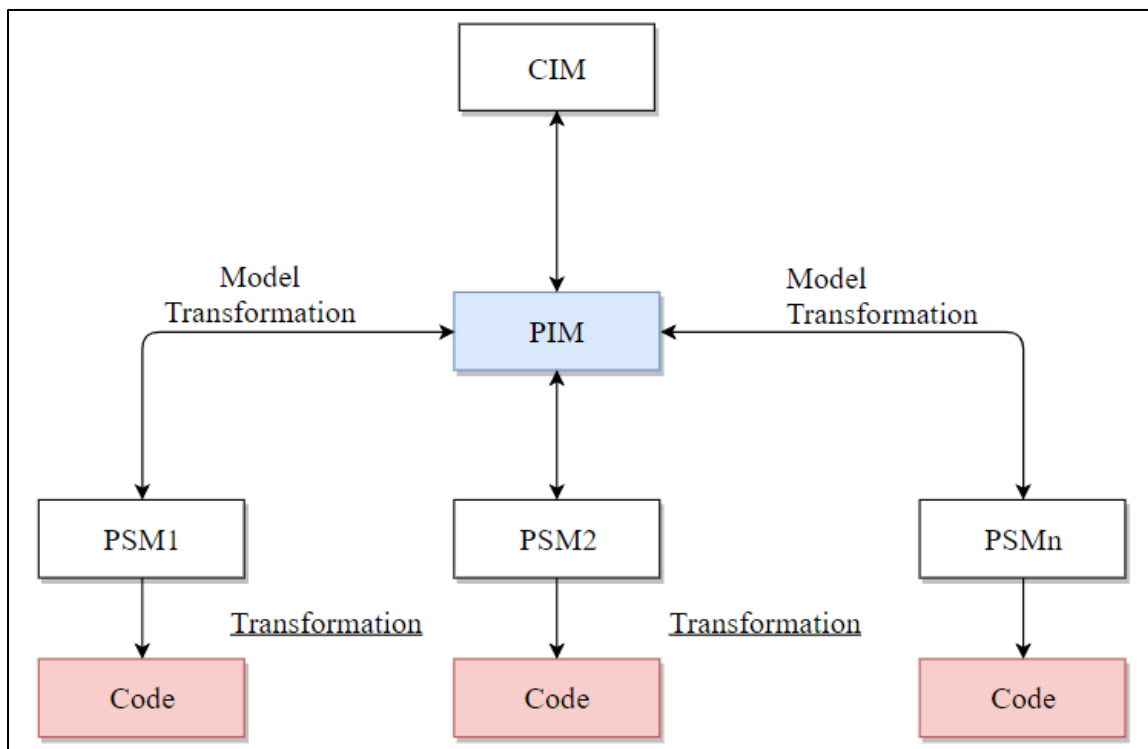


Figure 3.2: The Model Driven Architecture framework

Adopted from: (J. M. Morales, 2004)

Computation-Independent Model (CIM) is a model of the system and the environment in which it operates. It helps to describe the expected use of the system. Platform-Independent Model (PIM) models the system operation but abstracts the details of a specific platform. The Platform-Specific Model (PSM) is a model of the system in a particular platform specified by the PIM. Mapping rules are required for the transformation between PIM and PSM.

About this research, the standardized workflow interchange format borrows from the concept of a PIM whereas the PSM is a representation of the same workflow in formats specific to different WfMS. The interchange format that allows the transformation of the workflows has been discussed in Chapter 5. A workflow engine can implement several rules to govern the transformation of workflow from one WfMS to another. For instance, the geoprocesses can have different labels or names, yet they offer the same

functions in different WfMSs. Using mapping rules and ontology, the workflow engine can determine the corresponding process names to support the automatic transformation of the workflow.

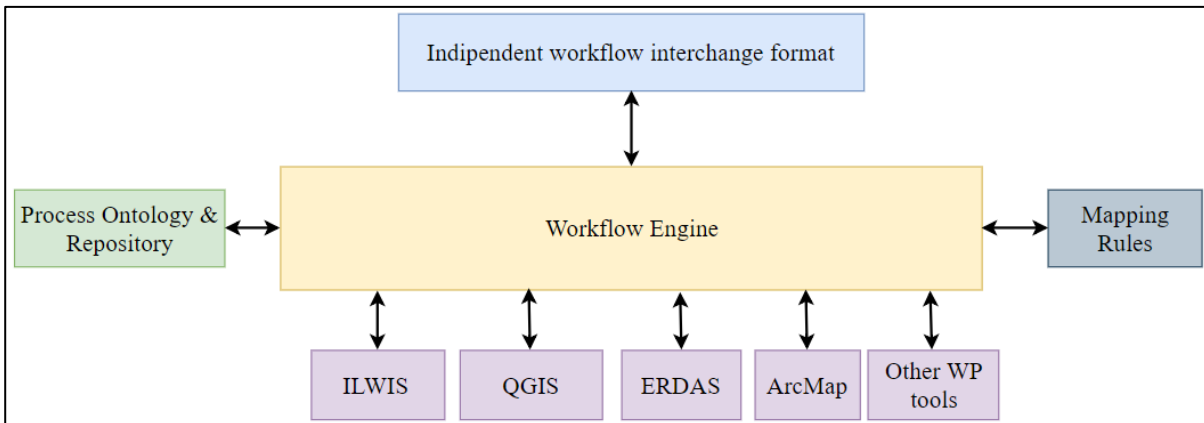


Figure 3.3: Architecture of Workflow Interchange formats

3.5.2. Towards a Web-based WfMS

Creating workflows is a task which requires considerable human effort and sharing them is often limited by undocumented and non-interoperable geoprocessing implementations (Lemmens, Toxopeus, et al., 2018). In as such, recreating the same workflow in different software packages becomes cumbersome. Most of the GIS software packages facilitate the sharing of data between them. They are capable of converting data from another vendor specific format to a format that can be understood by their software. However, the sharing of processes has not been handled by these software packages. Due to this, workflows have limited reusability outside the environment of their specific application software. Since scientific workflows are always created to solve a particular scientific problem, the sharing of these workflows across different software packages is becoming a necessity to allow scientist to share processes. Web-based platforms offer an opportunity for processes owned by specific WfMSs to be exposed as web services thereby increasing flexibility in the definition of the workflow and provide the extensible interface. Attempts to use web-based workflow management systems in scientific processes is not new. The OGC Testbed 13 (Pross & Christoph, 2018) demonstrated the applicability of BPEL and BPMN using Camunda modeler to compose and execute a shareable conflation workflow from OGC web services.

Moreover, there are also BPMN compliant web-based workflow clients that can be used in composing workflows from web services, for instance, the web-based JBPM editor, ProcessMaker and the BPMN modeler⁷. However, these WfMSs do not support reproduction of workflows from other non-standardization compliant WfMS thereby forcing users to recreate their workflow which requires considerable human effort. Furthermore, they do not provide a way in which users can visualize their inputs/outputs thus require the use of third-party software.

⁷ <https://demo.bpmn.io/>

This research proposes a web-based WfMS capable of integrating processes and data using web services as well as offers a method for sharing workflows between different WfMS. In the following Chapter, we discuss the composition of workflow from web services.

4. WORKFLOW COMPOSITION FROM DISTRIBUTED WEB SERVICES

In the previous chapter, we discussed the shortcomings of current WfMSs and proposed a web-based WfMS as a perfect solution that would make it possible to create workflows from web services and execute them using the underlying workflow engine. J. Morales & De By (2009) defined a web service as *an interface that describes a collection of operations that are network-accessible through standardized XML messaging*. However, the establishment of REST-based and light-weight exchange formats such as JSON also make it possible to access and reuse web services. We have discussed in the previously that JSON and REST services are gaining more acceptance today as compared to the traditional XML and SOAP-based services. Web services offer great potential for building service-oriented architectures thereby ensuring interoperability and accessibility of geoprocessing resources. Web services enable the use of multiple programming languages and utilities since service providers have different implementations for their services. These services are accessed using standards and APIs offered by service providers. The use of web services for distributed computing has increased tremendously in the past decade. This is because of the establishment of standards for managing the creation and sharing of geoprocessing resources. For instance, the OGC's WFS and WCS have made sharing of spatial data possible through the web. WFS makes it possible to share vector data using shapefiles or GeoJSON. Current GIS software is capable of reading and editing shapefiles and GeoJSON files. The WPS also allows consumption of remotely distributed processes. A large volume of scientific data is becoming available recently, and this has been attributed to the rise in production of high-resolution remote-sensing data and crowdsourcing technology which makes it possible to retrieve data faster and in high quantity (Yue et al., 2012).

Combining several web services in a workflow is seen as the new trend towards ensuring effective and efficient processing of real-time geospatial data. In this research, we define workflow composition as the process of aggregating or combining web services in which the output of one service is directed to the input interface of another service. The graphical tools, for instance, BPMN diagrams provide an intuitive means to specify workflows by linking web services graphically using nodes and connectors thereby offering a high-level abstraction of the underlying XML representation. The BPMN document representing the workflow is interoperable with most BPMN compliant WfMSs and allow reproduction of the visual workflow. Apart from BPMN, we discussed in the previous chapter that other interchange formats of workflow could be realized through JSON. This has already been implemented by several WfMSs such as ILWIS model builder, QGIS model, and ERDAS Spatial Modeler. However, these WfMSs do not incorporate web services making it impossible to use remote processes. As a result, users are forced to use locally available processes within their GIS software which are developed using the software developers' programming languages and utilities.

4.1. Composability of Scientific Workflows

Combining heterogeneous processing web services in a workflow can often pose problems to users because of their different requirements and implementations. For instance, the output data type of source operation can be different from the input data type of the target operation. In the case of geospatial data, different aspects of the data such as coordinate system and spatial resolution can lead to errors during the execution of the workflow. Diniz (2016) in his study of the composability of scientific workflows identified two types of errors that are associated with the incorrect composition of scientific workflows. The first category of errors is those that make the execution of the workflow to stop while the second category is often ignored but yield the wrong result. Some of these potential errors can be avoided during the composition stage of the workflow.

The process of checking if participating web services can work efficiently and give the desired result is known as workflow composability (Medjahed & Bouguettaya, 2005). Verification of composability of workflows from web services is currently not handled by most of the WfMSs to ensure correct sequencing and validation of incoming data. Figure 4.1: Levels of Composability of Scientific Workflows illustrates five levels of workflow composability examined by Diniz (2016) which includes structural composability, static syntactic composability, dynamic syntactic composability, semantic and qualitative composability. A WfMS should implement all the five levels of the workflow composability to yield a positive result.

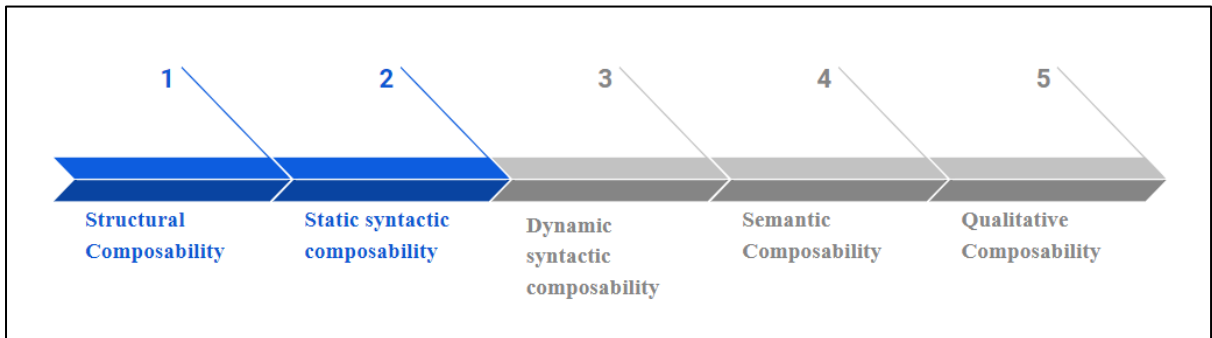


Figure 4.1: Levels of Composability of Scientific Workflows

Adopted from: Diniz (2016)

Structural composability ensures that the elements of the workflow composition are correctly connected. Nodes represent the processing services while the data flow between services is achieved using edges. In *static syntactic composability*, the output of a source service can only become an input to the target service if both of them are of the same data type. In case they are of different data types, then the output of the source services is automatically converted to the data type of the target service. *Dynamic syntactic composability* complements the previous static syntactic composability by ensuring that the outgoing and incoming data belong to the same type system which includes a coordinate system, geometrical dimension, temporal and spatial resolution. *Semantic composability* uses the semantic information of the data and process to ensure that they provide meaning and verifiable result. It makes use of the semantic web technology to derive the meaning of data and processes. Semantic composability makes it possible to substitute processes from a service ontology based on their semantic description. Ubels (2018) demonstrated the

same concept to convert abstract workflows to executable workflows. *Qualitative composability* evaluates user requirements against non-functional features such as response time, availability of the web service, cost, authorization and authentication, and legal rights. Because of the limitation of time and to avoid losing track of our objective, this research only implements structural and static syntactic levels of composability. However, we agree that other levels of composability are very important to the discovery of web services based on their semantic descriptions and whether they are exposed on the web.

4.2. Data Services

Composing workflows require data and processes which can be both offered as web services. OGC has defined several standards for sharing and reusing spatial data and processes. In this section, we look at some of the web services that offer a standardized interface to facilitate the creation and sharing of geospatial data on the internet.

4.2.1. Web Feature Service

The OGC Web Feature Service (WFS) offers methods for creating, modifying and retrieving vector format spatial data irrespective of the underlying data source. In this way, WFS provides an interface which can be used to retrieve the data without accessing the database or the source file. The WFS supports INSERT, UPDATE, DELETE, LOCK, QUERY and DISCOVERY operations on vector data using HTTP. The OGC specification for WFS defines several methods. For this research, we focused on three commonly used methods which include the *GetCapabilities*, *DescribeFeatureType*, and *GetFeature*. The *GetCapabilities* method is used to request a WFS server for the list of available operations and services. A *GetCapabilities* request can be issued using an HTTP GET or POST method. To make a successful *GetCapabilities* request, you need the URL of the WFS server, the name of service, request and the version of the WFS specification. A simple illustration is shown below.

WFS GetCapabilities ⁸	http://130.89.221.193:85/geoserver/ows? service=WFS& request=GetCapabilities& version=1.0.0
----------------------------------	--

The *GetCapabilities* request returns an XML response which can be explored to reveal available operations such as *DescribeFeatureType* and *FeatureTypeList* (List of features). From the *FeatureTypeList* we obtain the metadata information for a particular *FeatureType* (feature type). This information illustrated in Listing 4.1 includes the name, title, abstract or the description of the feature, keywords, and spatial reference information.

⁸ **Disclaimer:** The links provided in the examples may have been changed by the service providers.

Listing 4.1: WFS GetCapabilities Response

```

1. <FeatureType>
2.   <Name>triplesensor:citizen_points</Name>
3.   <Title>citizen_points</Title>
4.   <Abstract/>
5.   <Keywords>features, citizen_points</Keywords>
6.   <SRS>EPSG:4326</SRS>
7.   <LatLongBoundingBox minx="-3.09847" miny="11.143" maxx="-
   2.88357" maxy="11.265"/>
8. </FeatureType>
9. <FeatureType>
10.   <Name>maris_mamase:conservancies</Name>
11.   <Title>conservancies</Title>
12.   <Abstract/>
13.   <Keywords>features, conservancies</Keywords>
14.   <SRS>EPSG:21036</SRS>
15.   <LatLongBoundingBox minx="34.75756814360618" miny="-
   1.8350609541501903" maxx="35.82124917586963" maxy="-
   1.0417975842866478"/>
16. </FeatureType>
17. <FeatureType>

```

DescribeFeatureType is used to retrieve additional information about a particular feature type before actual data download. This method requires the URL of the WFS server, the name of the service, version of WFS specification, operation name, and the name of the feature type.

DescribeFeatureType	http://130.89.221.193:85/geoserver/wfs? request=DescribeFeatureType& version=1.0.0& TypeName= group1:waterbodies
---------------------	--

The *describefeaturetype* request returns an XML response (example Listing 4.2) showing metadata about the feature type such as the attributes of the data. In the response below, we can identify that the water bodies feature has three attributes which include location ID (LCID), landcover type (LANDCOVER) and the geometry (geom). The type of geometry is a multipolygon.

Listing 4.2: WFS DescribeFeatureType Response

```

1.
2. <xsd:schema xmlns:gml="http://www.opengis.net/gml" xmlns:group1="13
   0.89.221" xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormD
   efault="qualified" targetNamespace="130.89.221">
3. <xsd:import namespace="http://www.opengis.net/gml" schemaLocation="
   http://130.89.221.193:85/geoserver/schemas/gml/2.1.2/feature.xsd"/>
4. <xsd:complexType name="waterbodiesType">
5. <xsd:complexContent>
6. <xsd:extension base="gml:AbstractFeatureType">
7. <xsd:sequence>

```

```

8. <xsd:element maxOccurs="1" minOccurs="0" name="LCID" nillable="true"
   type="xsd:string"/>
9. <xsd:element maxOccurs="1" minOccurs="0" name="LANDCOVER" nillable="
   true" type="xsd:string"/>
10. <xsd:element maxOccurs="1" minOccurs="0" name="geom" nillable
    ="true" type="gml:MultiPolygonPropertyType"/>
11. </xsd:sequence>
12. </xsd:extension>
13. </xsd:complexContent>
14. </xsd:complexType>
15. <xsd:element name="waterbodies" substitutionGroup="gml:_Featu
    re" type="group1:waterbodiesType"/>
16. </xsd:schema>

```

The *GetFeature* operation returns the selection of features from the data source. This method allows one to specify the output data format. For this research, a GeoJSON data format is preferred because it is light-weight and can be easily adopted by the applications. The OGC RESTful services by default provide GeoJSON data format for GetFeature request.

SOAP	GET http://130.89.221.193:85/geoserver/wfs? request=GetFeature& version=1.0.0& TypeName=group1:waterbodies& Outputformat=application/json
REST	GET http://130.89.221.193:85/geoserver/wfs/1.0.0/group1:waterbodies/ Accept : application/vnd.geo+json

4.2.2. Web Coverage Service

The OGC Web Coverage Service (WCS) is a standard that is used to retrieve raster data or coverages from a geospatial server. This service uses *GetCoverage* operation to access raster data and request metadata about the raster data through the *DescribeCoverage* operation. The *GetCapabilities* of the WCS performs the same function as that of the WFS. However, it retrieves a list of valid WCS operations and services. Listing 4.3 shows the contents of a WCS *GetCapabilities* XML response which includes a coverage ID (*ows:CoverageId*) and the bounding box (*ows:BoundingBox*).

GetCapabilities	http://130.89.8.26:85/geoserver/ows? service =WCS& request =GetCapabilities
-----------------	---

Listing 4.3: WCS GetCapabilities Response

```

1. <wcs:Contents>
2. <wcs:CoverageSummary>...</wcs:CoverageSummary>
3. <wcs:CoverageSummary>...</wcs:CoverageSummary>
4. <wcs:CoverageSummary>...</wcs:CoverageSummary>
5. <wcs:CoverageSummary>...</wcs:CoverageSummary>
6. <wcs:CoverageSummary>...</wcs:CoverageSummary>
7. <wcs:CoverageSummary>...</wcs:CoverageSummary>
8. <wcs:CoverageSummary>
9.   <wcs:CoverageId>maris_mamase__carcap_kg_23m</wcs:CoverageId>
10.    <wcs:CoverageSubtype>RectifiedGridCoverage</wcs:CoverageSubty
    pe>
11.    <ows:WGS84BoundingBox>
12.      <ows:LowerCorner>34.763885106614055 -
13.      1.8323458961775405</ows:LowerCorner>
14.      <ows:UpperCorner>35.82010641618325 -
15.      1.0380577375481044</ows:UpperCorner>
16.    </ows:WGS84BoundingBox>
17.    <ows:BoundingBox crs="http://www.opengis.net/def/crs/EP
    SG/0/null">
18.      <ows:LowerCorner>696275.4 9797373.3700000003</ows:LowerCorner>
19.      <ows:UpperCorner>813775.4 9885123.3700000003</ows:UpperCorner>
20.    </ows:BoundingBox>
    </wcs:CoverageSummary>
    </wcs:CoverageSummary>

```

The *DescribeCoverage* operation is important when more information about the coverage or raster data is required. The additional information provided by the *DescribeCoverage* operation includes information about the coordinate reference system, metadata about the coverage, the domain, range and formats for available for retrieving the data.

DescribeCoverage	http://130.89.8.26:85/geoserver/ows? service=WCS& request=DescribeCoverage& coverageid=maris_mamase:carcap_kg_23m& version=1.0.0
------------------	---

The *GetCoverage* operation facilitates the acquisition of the raw raster data from the WCS server. The interoperable data format used in this research to retrieve coverages is the GeoTIFF. This is because GeoTIFF files can be read by most GIS software.

GetCoverage	http://130.89.8.26:85/geoserver/ows?version=2.0.0& service=WCS& request=GetCoverage& coverageid=maris_mamase:DMintake_kg_23m_nrdays& format=image/geotiff
-------------	---

4.2.3. Sensor Web Enablement

Pervasive and ubiquitous computing are computer science concepts that describe the growing trend of embedded computational capabilities. Most of today's electronic devices are equipped with microcontrollers which provides them with abilities to sense their environment and communicate with each other through the internet. In what is popularly known as the Internet of Things (IoT), many devices today are composed of a large number of sensor nodes which collect information about their surroundings. Some of this information includes temperature, rainfall, river water levels, light intensity, air composition, GPS locations among others. These pieces of information which are provided in real-time can be useful for monitoring pollution, managing disaster, weather forecasting, managing natural resources, etc. Sharing and accessing real-time information provided by the numerous sensors require a collection of web-based services to maintain the registry of sensors and the type of information which they transmit. However, to achieve interoperability for cross-organization activities, a web technology standard for describing sensors, their outputs, control parameters and location should be considered (Chu, Kobialka, Durnota, & Buyya, 2006).

The OGC (2012) established the Sensor Web Enablement (SWE) which provides a suite of standards specifying protocols for discovery, access, and sharing of sensor data. Rouached et al. (2012) categorized SWE framework into two categories where the *interface model* defines the standards for sensor related web services and the *information model* defines the standards that offer specification for sensor data formats.

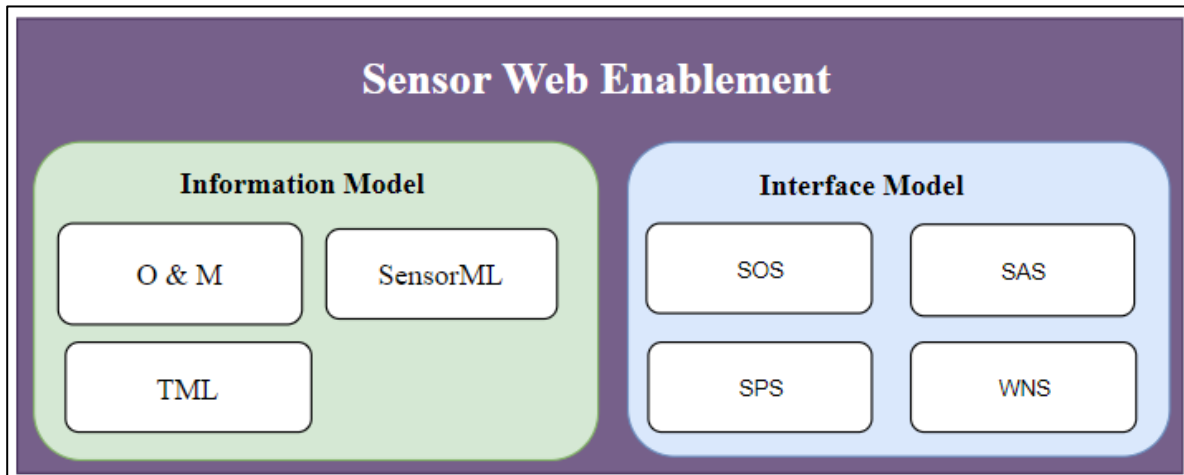


Figure 4.2: Sensor Web Enablement Framework

Source: Rouached et al. (2012)

SWE Information Model

The SWE information model services provide schemas for XML encoding of observations, and measurements as well as a description for sensor platforms regarding discovery, query, and control of sensors. These services include the Sensor Model Language (SensorML), Observation & Measurement (O & M) and Transducer Markup Language (TML).

SWE Interface Model

This model provides a specification of interfaces for different sensor web services. These include the Sensor Observation Service (SOS), Sensor Alert Service (SAS), Sensor Planning Service (SPS) and Web

Notification Service (WNS). Sensor Observation Service was designed to offer uniform access to observations from different sensors, and it enables querying and updating of sensor data and metadata (Bröring et al., 2011). The Sensor Alert Service, on the other hand, offers a notification service by pushing sensor data to subscribed users based on a defined criterion. SOS is a pull-based service whereas SAS is a push-based service. SPS is used for setting sensor parameters and enables the tasking of sensors. WNS is used to manage sessions between the clients and the SWE services using asynchronous notification mechanism.

SOS implements three operations similar to WFS and WCS regarding their functions. The *GetCapabilities* is used to extract metadata information for the sensor data. Searching and retrieving observations is handled by the SOS *GetObservation* operation whereas the *DescribeSensor* operation is used to query particular sensor descriptions. The following properties are always associated with these operations.

Procedure: This is the sensor, instrument, method or algorithm used to make the observation.

Offering: This is a collection of observations produced by one sensor. For instance, a sensor may observe temperature, water level, humidity, etc.

Observed property: This is a particular item referenced by its name which is observed by the sensor, e.g., temperature.

Feature of Interest: This is a pointer to a specific feature of interest.

GetCapabilities	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=GetCapabilities& service=SOS
DescribeSensor	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=DescribeSensor& service=SOS& procedure= Lufttemperatur-Frankfurt_Osthafen_24700404& outputformat=text/xml;subtype='sensorml/1.0.1'& version=1.0.0
GetObservation ⁹	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=GetObservation& service=SOS& procedure=Lufttemperatur-Frankfurt_Osthafen_24700404& version=1.0.0& offering=LUFTTEMPERATUR& observedProperty=Lufttemperatur&

⁹ **Disclaimer:** The links provided in the examples may have been changed by the service providers.

	featureOfInterest =Frankfurt_Osthafen_24700404& responseformat =text/xml;subtype="om/1.0.0"
--	--

Integration of SWE with the Human Sensor Web to handle human generated contents which includes textual descriptions human collected observations by sensors held by humans is a current research topic for users of crowdsourced geoinformation (Bröring et al., 2011). Combining satellite data with the sensor and crowdsourced geoinformation obtained through web services provide a means through which real-time analysis of natural phenomenon can be achieved. For instance, the ongoing project of AfriAlliance which is our case study makes use of human sensors and the traditional in-situ and satellite data to monitor and forecast water resources (Mannaerts et al., 2017b). The OGC SOS was found to be relevant for this research since it provides operations for retrieving sensor observations.

4.3. Processing Services

The web service technology and the advent of cloud computing have made it possible for software application vendors to host geoprocessing services in the cloud. The different levels of cloud computing provision such Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) makes separation of concerns very possible in the geoprocessing web. Web services allow real-time processing of geospatial data for creating value-added information. Exposing geoprocessing functions in the cloud computing platforms has the benefits of bringing scalable, reliable and cost-effective processing services to users (Yue et al., 2012). Remotely available processing services are essential building blocks for geoprocessing workflows. The increasing number of distributed and heterogeneous processing web services has motivated the establishment of a standard to facilitate publication and access to remote geospatial processing services. For instance, the OGC WPS was established in 2007 to provide rules for defining geoprocessing web services. However, this standard has not been adopted by several providers of computing services because of its over-reliance on SOAP technology which is being considered out of fashion today. Instead, they rely on their implementations using REST technology. The OGC Testbed-13 introduced a WPS implementation using REST architecture in a bid to win the growing REST community (Gonçalves, 2017). However, this new development does not provide instructions on how to achieve process chaining. Moreover, it has not yet been adopted as a standard. In the following subsections, we discuss the implementation of OGC WPS and non-OGC compliant processing services.

4.3.1. OGC Web Processing Service

The OGC Web Processing Service offers a standardized interface for publishing of geospatial processes, algorithms, and calculations. This service offers three key operations for interacting with remote processes which are mainly the *GetCapabilities*, *DescribeProcess* and the *Execute*. The *GetCapabilities* just like other implementations for WFS and WCS is used to request metadata information for processes available in a WPS server.

SOAP	GET http://130.89.221.193:85/geoserver/ows? service=WPS& request=GetCapabilities
REST	GET http://geoprocessing.demo.52north.org:8080/wps-proxy

The SOAP WPS GetCapabilities returns an XML response shown in Listing 4.4 which when explored shows a list of services and operations available. The information about a process includes the process identifier, title, and description (abstract). In contra, the RESTful WPS GetCapabilities request returns a JSON object shown in Listing 4.5.

Listing 4.4: WPS GetCapabilities response using SOAP bindings

1.	<wps:ProcessOfferings>
2.	<wps:Process wps:processVersion="1.0.0">... </wps:Process>
3.	<wps:Process wps:processVersion="1.0.0">... </wps:Process>
4.	<wps:Process wps:processVersion="1.0.0">... </wps:Process>
5.	<wps:Process wps:processVersion="1.0.0">
6.	<ows:Identifier> JTS:centroid </ows:Identifier>
7.	<ows:Title> Centroid </ows:Title>
8.	<ows:Abstract>
9.	Returns the geometric centroid of a geometry. Output is a single point. The centroid point may be located outside the geometry.
10.	</ows:Abstract>
11.	</wps:Process>
12.	<wps:Process wps:processVersion="1.0.0">
13.	<ows:Identifier> JTS:contains </ows:Identifier>
14.	<ows:Title> Contains Test </ows:Title>
15.	<ows:Abstract>
16.	Tests if no points of the second geometry lie in the exterior of the first geometry and at least one point of the interior of second geometry lies in the interior of first geometry.
17.	</ows:Abstract>
18.	</wps:Process>
19.	<wps:Process wps:processVersion="1.0.0">... </wps:Process>
20.	<wps:Process wps:processVersion="1.0.0">... </wps:Process>

The observable keywords of the JSON object include *ProcessSummaries* which is an array of processes. Each process has a unique identifier, title, process version, url among others. In contra to the result in Listing 4.4, the RESTful WPS GetCapabilities responses miss the description of a process.

Listing 4.5: RESTful WPS GetCapabilities Response

```

1  {
2    "Capabilities": {
3      "ServiceIdentification": {},
14     "ServiceProvider": {},
24     "Contents": {
25       "ProcessSummaries": [
26         {
27           "identifier": "org.n52.wps.extension.QuakemapAlgorithm",
28           "title": "Quakemap algorithm",
29           "_processVersion": "1.1",
30           "_jobControlOptions": "sync-execute async-execute",
31           "_outputTransmission": "value reference",
32           "url": "http://geoprocessing.demo.52north.org:8080/wps-proxy/
           processes/org.n52.wps.extension.QuakemapAlgorithm"
33         },
34         {},
42         {},
50         {},
58         {},
66         {},
74         {},
82         {},
90         {},
98         {},
106        {},
114        {}
122       ]
123     },
124     "_service": "WPS",
125     "_version": "2.0.0"
126   }
127 }

```

The *DescribeProcess* operation provides more information about a particular process. These include input requirements, allowable formats and the output information for a process.

SOAP	GET http://130.89.221.193:85/geoserver/ows? service=WPS& request=DescribeProcess& identifier=JTS:centroid
REST	GET http://geoprocessing.demo.52north.org:8080/wps- proxy/processes/org.n52.wps.server.algorithm.JTSConvexHullAlgorithm

The identifier parameter specifies the identity of the processes to describe. An example of the result of the DescribeProcess operation is shown in Listing 4.6 and contains information about the process including *ows:Identifier*, *ows:Title*, *ows:Abstract* (description), *DataInputs* and *ProcessOutputs*. The input and output parameters specifications included the identifier, title, abstract, multiplicity and supported data type and format.

Listing 4.6: WPS DescribeProcess using SOAP Bindings

```

1. <wps:ProcessDescriptions xmlns:xs="http://www.w3.org/2001/XMLSchema
   " xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.
   opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" x
   mlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xml:lang="en" service="WPS" version="1.0.0" xsi:schemaLoc
   ation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/
   wps/1.0.0/wpsAll.xsd">
2. <ProcessDescription wps:processVersion="1.0.0" statusSupported="tru
   e" storeSupported="true">

```

```

3. <ows:Identifier>JTS:centroid</ows:Identifier>
4. <ows:Title>Centroid</ows:Title>
5. <ows:Abstract>
6. Returns the geometric centroid of a geometry. Output is a single po
   int. The centroid point may be located outside the geometry.
7. </ows:Abstract>
8. <DataInputs>
9. <Input maxOccurs="1" minOccurs="1">
10.   <ows:Identifier>geom</ows:Identifier>
11.   <ows:Title>geom</ows:Title>
12.   <ows:Abstract>Input geometry</ows:Abstract>
13.   <ComplexData>
14.   <Default>
15.   <Format>
16.   <MimeType>text/xml; subtype=gml/3.1.1</MimeType>
17.   </Format>
18.   </Default>
19.   <Supported>...</Supported>
20.   </ComplexData>
21. </Input>
22. </DataInputs>
23. <ProcessOutputs>
24. <Output>...</Output>
25. </ProcessOutputs>
26. </ProcessDescription>
27. </wps:ProcessDescriptions>

```

The RESTful WPS DescribeProcess request provides a JSON object shown in Listing 4.7. We observed similar keywords to the result of the SOAP-based request which included *Title*, *Identifier*, *Input*, *ComplexData* and *Output*.

Listing 4.7: RESTful WPS DescribeProcess Result

```

1 {
2   "ProcessOffering": {
3     "Process": {
4       "Title": "org.n52.wps.server.algorithm.JTSConvexHullAlgorithm",
5       "Identifier": "org.n52.wps.server.algorithm.JTSConvexHullAlgorithm",
6       "Input": [
7         {
8           "Title": "data",
9           "Identifier": "data",
10          "ComplexData": {
11            "Format": [""]
12          },
13          "_minOccurs": "1",
14          "_maxOccurs": "1"
15        }
16      ],
17      "Output": [
18        {
19          "Title": "result",
20          "Identifier": "result",
21          "ComplexData": {
22            "Format": [""]
23          }
24        }
25      ]
26    },
27    "_processVersion": "1.1.0",
28    "_jobControlOptions": "sync-execute async-execute",
29    "_outputTransmission": "value reference",
30    "execute-url": "http://geoprocessing.demo.52north.org:8080/wps-proxy/processes/org.n52.wps.server.algorithm.JTSConvexHullAlgorithm/jobs"
31  }
32 }

```

The *Execute* operation is a request to run the specified process with the supplied input parameters to produce the required data outputs. Since the WPS execute request is complex, it is always sent in the body

of a POST form. WPS execute allows passing data by value or by reference using WFS or WCS. In the example shown in Listing 4.8 below, the value of the input data has been specified by reference using WFS. An example of a RESTful WPS Execute request body where data is passed by value is shown in Listing 4.9. Passing of data by reference is more preferred for this research as compared to by value since it can be achieved by the OGC WFS and SOS.

Listing 4.8: WPS Execute Request's Body for SOAP Binding

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://www.w3
   .org/2001/XMLSchema-
   instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http:
   //www.opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0"
   xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.o
   pengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="h
   ttp://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://www.w3.org/199
   9/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http:
   //schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
3.   <ows:Identifier>gs:Centroid</ows:Identifier>
4.   <wps:DataInputs>
5.     <wps:Input>
6.       <ows:Identifier>features</ows:Identifier>
7.       <wps:Reference mimeType="application/json" xlink:href="
   http://localhost:8585/geoserver/ows/wfs?SERVICE=WFS&VERSION=1.0.0&R
   EQUEST=GetFeature&TYPENAME=topp:tasmania_state_boundaries&OUTPUTFOR
   MAT=application/json" method="GET"/>
8.     </wps:Input>
9.   </wps:DataInputs>
10.  <wps:ResponseForm>
11.    <wps:RawDataOutput mimeType="application/json">
12.      <ows:Identifier>result</ows:Identifier>
13.    </wps:RawDataOutput>
14.  </wps:ResponseForm>
15. </wps:Execute>

```

Listing 4.9: WPS Execute Request's Body for RESTful Binding

```

1 {
2   "Execute": {
3     "Identifier": "org.n52.wps.server.algorithm.JTSConvexHullAlgorithm",
4     "Input": [
5       {
6         "ComplexData": {
7           "_mimeType": "application/wkt",
8           "_text": "POLYGON((847666.55940505 6793166.084248,849319
          .51014149 6793452.723104,848402.26580219 6792640
          .5796786,849873.67859648 6792439.9324794,847666
          .55940505 6793166.084248))"
6         },
7         "_id": "data"
8       }
9     ],
10    "output": [{
11      "_mimeType": "application/vnd.geo+json",
12      "_id": "result",
13      "_transmission": "value"
14    }],
15    "_service": "WPS",
16    "_version": "2.0.0"
17  }
18 }

```

4.3.2. Non-OGC Compliant Processing Services

Several geoprocessing services are also available which do not follow the standards of the OGC WPS. Workflow should incorporate such services to give users opportunities to use a variety of services in their workflows. Example of non-OGC compliant processing services includes the RESTful coordinate transformation service that takes an input point and transforms it from the source to the target coordinates system.

<http://gip.itc.nl/services/wcts.py?>

coords=6.5823,52.1487&

sourcecrs=4326&

targetcrs=28992

<http://gip.itc.nl/services/coordinatetransform/6.5823,52.1487/4326/28992>

The specification of the above coordinate transformation can be described in the following manner using the OGC WPS.

REST endpoint¹⁰

The REST endpoint for this service is <http://gip.itc.nl/services/coordinatetransform>.

Input Metadata	Value
identifier: coords	6.5823,52.1487
abstract: Input Coordinate	
data type: vector geometry (point)	
optional: false	

¹⁰ **Disclaimer:** The links provided in the examples may have been changed by the service providers.

identifier: sourcecrs 4326

abstract: Source SRS

data type: integer

optional: false

identifier: targetcrs 28992

abstract: Target SRS

data type: integer

optional: false

Output Metadata	Value
identifier: result	
abstract: Result of Coordinate transformation	
data type: vector geometry (point)	

4.4. OGC Process Chaining

The OGC process chaining is a concept where two or more WPS are combined into a single powerful WPS. The process chaining behaves more like function calling in programming where one function's output becomes an input to another function. Chaining processes is a useful feature of WPS which enables the creation of complex workflows from web services (Meek et al., 2016). The OGC WPS standard 1.0 Open Geospatial Consortium (2012) recommends three ways in which process chaining can be achieved.

1. By use of BPEL engine to orchestrate services.
2. By designing a WPS that calls other WPS processes in a sequence.
3. By cascading services in a chain as part of the execute request.

The use of a BPEL engine to orchestrate services allows the workflow engine to monitor and manipulate services. BPEL engines are capable of converting a BPMN document to an executable BPEL script. However, the use of BPEL has brought discussion among the scientific community since BPEL does not have a standardized graphical notation to represent workflows and relies heavily on BPMN (Meek et al., 2016). This has influenced the scientist to prefer using BPMN engines to execute workflows. BPEL scripts are based on SOAP architecture which is becoming less popular nowadays because of the increasing adoption of RESTful services. The second approach to achieve process chaining by calling processes within a WPS seems a perfect solution however it cannot be used for RESTful processing services because the current WPS specifications only implement process chaining for WPS SOAP bindings. The second approach also works only when the processing services are being offered by a single WPS server thereby limiting opportunities for distributed processing. This research adopts the third

approach to chaining processes within an execute request because it provides more flexibility for incorporating OGC WPS and non-OGC RESTful processing services.

4.5. Workflow Engine

The composition of web services only yields a visual and textual representation of the workflow. To execute the workflow and produce some meaningful result, a workflow engine is required. The Workflow Management Coalition (WfMC), Hollingsworth (1995), defines a workflow engine as *a software service that provides the runtime execution environment for a workflow instance*. Workflow engines are responsible for coordination of the execution process of the entire workflow by ensuring that data flows sequentially to linked processes. Workflow engines should be able to handle errors that occur during the execution of the workflow and convey a message with the error information to the user. A workflow engine that gives users the ability to orchestrate and execute distributed web services is desirable to support scientific research.

The WfMC's workflow architecture identifies major components and five interfaces to these components as demonstrated in Figure 4.3: The WfMC Workflow Architecture. The process definition tools consist of all the processes available for composing a workflow. In the case of this research, the process definition tools represent a list of geoprocesses which have been exposed as web services. The workflow enactment service consists of one or more workflow engines where management and execution of workflow instance take place. The Workflow API provides an interface where workflow clients can make requests to the workflow engine which include execution of a defined workflow, passing of relevant data and transformation of workflow interchange formats of different WfMSs.

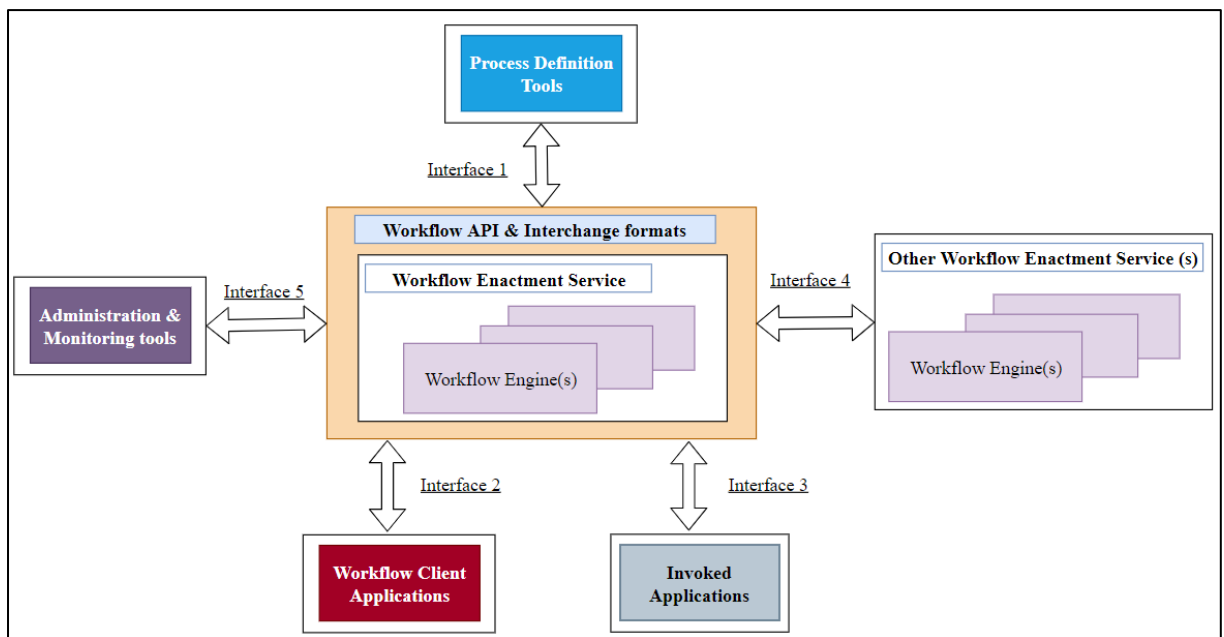


Figure 4.3: The WfMC Workflow Architecture

The WfMSs which were discussed in the previous chapter have their workflow engines which can execute their workflow specifications. For instance, an ILWIS workflow can only be executed by an ILWIS engine. Likewise, a BPMN document can only be executed using a BPMN engine. This is a limitation to the current WfMSs since the execution of external processes cannot be achieved. This limitation of current WfMSs motivated the implementation of our workflow engine. Our workflow engine performs the following functionalities some of which cannot be achieved by the current WfMSs.

- i. Translate the JSON representation of the workflow to an executable script which can be executed by the workflow engine. This involves automatic creation of WPS execute body using XML for OGC compliant WPS.
- ii. Control and coordinate the execution of the workflow by orchestrating web services according to the order of the service composition.
- iii. Generate downloadable results and provide users with the ability to view their result as a Web Mapping Service (WMS).
- iv. Transform workflow produced by one WfMS to another WfMS.

5. SUPPORTING SHAREABILITY AND REPRODUCIBILITY OF WORKFLOWS

In the previous chapters, we discussed the current WfMSs and the challenges they face to support the shareability and reproducibility of workflows. It was realized that to make workflows shareable and reproducible; a standard workflow interchange format is required which can facilitate the transformation of workflows from one platform to the other. We also proposed a web-based WfMS with a client component allowing users to create and edit workflows, and a workflow engine that can execute and transform workflows from different WfMSs. This chapter proceeds with the discussion on the methodology to adopt in transforming workflows from one WfMS to another to realize shareability and reproducibility. Kechagioglou & Lemmens (2018) identified two important considerations to ensure successful transformation of workflows from one WfMS to another. The first consideration which supports shareability involves the conversion of notation specific constructs from the source software through an intermediate then to the target WfMS while still maintaining the semantics and data flow in the workflow. WfMSs should provide sufficient constructs and metadata for expressing the flow logic in their workflows. The second consideration is responsible for the reproducibility of the workflow. It involves the detection of corresponding operators, for instance, internal names of processes in different WfMSs making it easy to reconstruct the workflow based on the input and output requirements of the target WfMS's operation. A study by Ubels (2018) found out that corresponding operators of the target WfMS and their notations can be obtained using Semantic Web technology, operations ontology, and Linked Data. Though this was demonstrated with a proof of concept using ArcGIS desktop application, his concept can still be useful for web services provided the software vendors to expose APIs for their various GIS software.

5.1. Supporting Shareability through Standard Interchange Format

5.1.1. Mapping Workflow Constructs of different WfMSs

As was observed in Chapter 3, most GIS software packages have adopted JSON as the primary data format for representation of their workflows. However, it was found that all the GI software packages have their schema for their workflow interchange format. As a result, there is a desire to come up with a standardized interchange format for scientific workflows to achieve interoperability. Mendling et al. (2004) observed that for universal interchange format to be successful, it should reflect at least the commonly used grammar among different software packages. In that regard, this research attempted to consolidate the schema of workflow interchange formats of various GI software packages which are already discussed. Additionally, it also looked at the XML-based BPMN documents and OGC Geoprocessing Workflow supported by WPS, to identify common constructs for the JSON schema.

The mapping between elements of BPMN, OGC GPW, ILWIS, and QGIS was obtained based on the schema and structure of the workflow exchange formats that were discussed in Section 3.1 and Section 3.2. For example, Table 5.1 below shows that a BPMN *serviceTask* element can have a corresponding name of the operation in OGC GPW and ILWIS, and algs in QGIS. However, there are certain keywords which did not have a one-to-one match because of the different implementation of the corresponding WfMSs.

These findings correspond to the work done by Kechagioglou & Lemmens (2018) where they observed the relationship between BPMN document and ILWIS workflow. There are however some differences in this research as compared to their study. In their study, they used the BPMN *scriptTask* element to map to the *operation* element of ILWIS. However, this research is based on using web services, and therefore we adopt *serviceTask* instead of *scriptTask*. This research has also identified additional elements such as *id*, *inputSet*, *outputSet*, and *implementation*.

Table 5.1: Mapping Workflow Elements for different Workflow Specifications

BPMN	OGC GPW	ILWIS	QGIS
process	process	workflow	values
serviceTask	operation	operation	algs
inputSet	DataInputs	inputs	inputs
outputSet	ResponseForm	outputs	outputs
dataInput	Input	input	params
dataOutput	RawDataOutput	output	
sequenceFlow		connection	
incoming	receive	fromOperationID	ValueFromOutput
outgoing	reply	toOperationID	
id	identifier	longname	consoleName
implementation		resource	

5.1.2. Standard JSON Schema for Sharing Workflows

Based on the findings in the previous section, the JSON schema for standard interchange format was recommended to have the four main components which include an identifier, metadata, operations, and connections. Figure 5.1 illustrates the relationship between a workflow and its components. A workflow must contain metadata and one or many operations. Moreover, a workflow must also contain a definition of the connections between its operations. An operation, on the other hand, can be connected to another operation (s). An operation has metadata which describes it as a processing resource. It also contains one or more input (s) and output (s). It can also be observed that a workflow can be an operation. The attributes of the diagram are discussed in the following paragraphs and a concrete class diagram which describes the whole workflow and its elements is shown later in Figure 5.2.

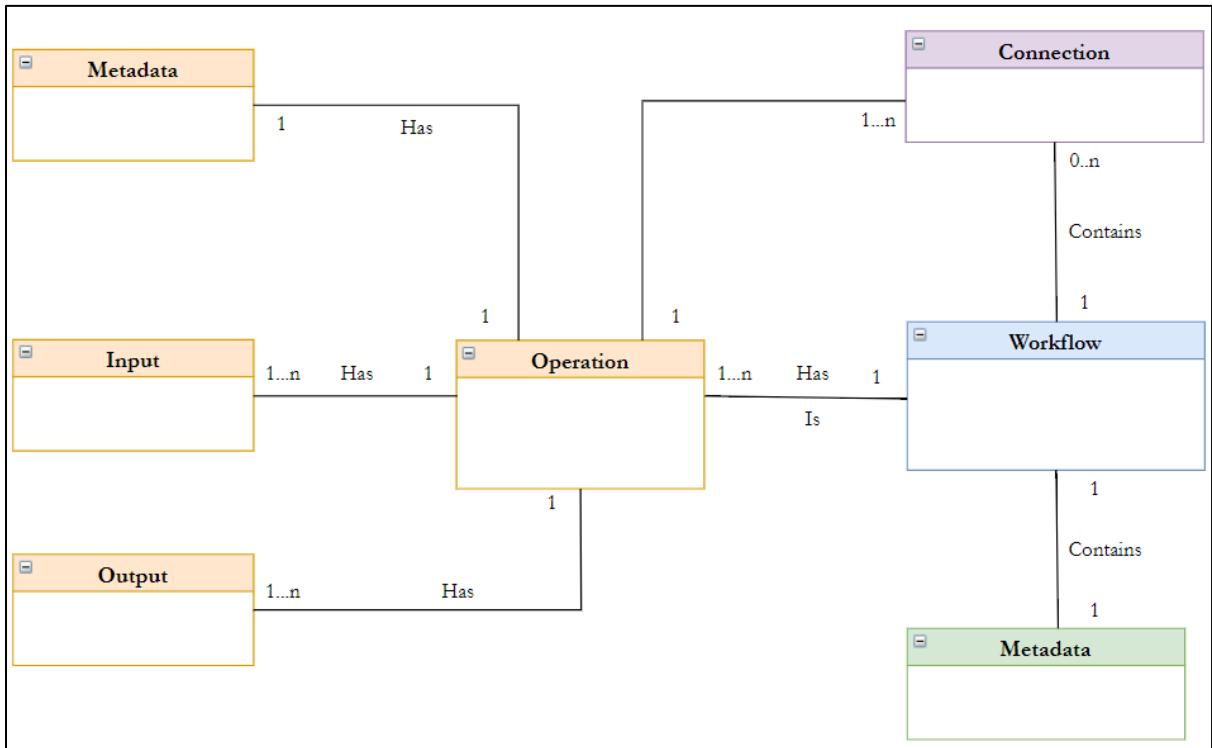


Figure 5.1: Abstract Class diagram for a Workflow

A. Identifier

This is a unique number which is used to refer to a particular workflow. The keyword used for this is “id,” and the value is of the integer data type.

B. Metadata

This object contains the descriptive details of the workflow which includes the name of the workflow and description. The description of the workflow entails what the function of the workflow itself. The produced JSON schema for the metadata property is shown in Listing 5.1.

- i. longname: String data type.
- ii. description: String data type.

Listing 5.1: JSON schema for workflow metadata property

```

1. {
2.   "$schema": "http://json-schema.org/draft-04/schema#",
3.   "type": "object",
4.   "properties": {
5.     "metadata": {
6.       "type": "object",
7.       "properties": {
8.         "longname": {
9.           "type": "string"
10.        }
11.      },
12.      "required": [
13.        "longname"

```

```

14.         ]
15.     }
16. },
17.     "required": [
18.         "metadata"
19.     ]
20. }

```

C. Operations

A workflow can have one or more operations. An operation is used synonymously to mean a process or activity in the workflow. The operations of a workflow are contained in an array object having the keyword “operations.” The general properties of every operation are as shown in the table below. A JSON scheme was produced from these properties which are illustrated in Listing 5.2.

Property	Description
id	Integer type specifying the index of the operation in the operations array.
metadata	JSON object providing descriptive information of the operation
inputs	An array of input data.
outputs	An array of output data.

Listing 5.2: JSON schema for the properties of an operation

```

1. {
2.   "$schema": "http://json-schema.org/draft-04/schema#",
3.   "type": "object",
4.   "properties": {
5.     "id": {
6.       "type": "integer"
7.     },
8.     "metadata": {
9.       "type": "object"
10.    },
11.    "inputs": {
12.      "type": "array",
13.      "items": {}
14.    },
15.    "outputs": {
16.      "type": "array",
17.      "items": {}
18.    }
19.  },
20.  "required": [
21.    "id",
22.    "metadata",
23.    "inputs",
24.    "outputs"
25.  ]
26. }

```

a.) Identifier

The identifier of an operation is a unique integer number that is used to show the index of an operation in the workflow. This number is generated by the workflow editor and is not part of the metadata of an operation.

b.) Metadata

The metadata object of an operation contains properties which are obtained from the metadata definition of the process. For instance, the OGC WPS Get Capabilities requests, identifies that a process has the following metadata definitions.

Property	Description
ows:Identifier	An identifier which uniquely identifies a process in the WPS server.
ows:Title	It refers to the long name of the process.
ows:Abstract	The description of the WPS process.

The metadata of QGIS algorithms also contains some of the following properties:

Property	Description
name	Name of the algorithm.
consoleName	It refers to the internal name of the algorithm.
description	Description of the algorithm.
pos	The position of the visual object for the algorithm specifying the X and Y screen position.

The standardized JSON interchange format consolidates the metadata schema of the selected GI software packages (ILWIS and QGIS), and the OGC WPS Get Capabilities is illustrated below. The corresponding JSON schema for the metadata property is shown in Listing 5.3.

Property	Description
longname	Long name of the operation.
label	A label is representing the internal name of the operation. This uniquely identifies the operation. Operations can have the same long name but different labels.
description	Description of the operation and what it does.
inputparametercount	Input parameter count is referring to the number of inputs that an operation can have.
outputparametercount	Output parameter count is referring to the number of outputs that an operation can yield.

url	Uniform resource locator (URL) referring to the address of the operation endpoint.
resource	Resource identifies the owner of the process; for instance, a WPS or an ILWIS process.
position	The position of the visual object for the process specifying the X and Y screen coordinates

Listing 5.3: JSON schema for the operation's metadata property

```

1. {
2.   "$schema": "http://json-schema.org/draft-04/schema#",
3.   "type": "object",
4.   "properties": {
5.     "metadata": {
6.       "type": "object",
7.       "properties": {
8.         "longname": {
9.           "type": "string"
10.        },
11.        "label": {
12.          "type": "string"
13.        },
14.        "url": {
15.          "type": "string"
16.        },
17.        "resource": {
18.          "type": "string"
19.        },
20.        "description": {
21.          "type": "string"
22.        },
23.        "inputparametercount": {
24.          "type": "integer"
25.        },
26.        "outputparametercount": {
27.          "type": "integer"
28.        },
29.        "position": {
30.          "type": "array",
31.          "items": [
32.            {
33.              "type": "integer"
34.            },
35.            {
36.              "type": "integer"
37.            }
38.          ]
39.        },
40.        "required": [
41.          "longname",
42.          "label",
43.          "url",
44.          "resource",
45.          "description",

```

```

47.         "inputparametercount",
48.         "outputparametercount",
49.         "position"
50.     ]
51. }
52. },
53.     "required": [
54.         "metadata"
55.     ]
56. }

```

c.) Inputs

An operation can have one or more inputs. The inputs of an operation are therefore contained in an array object. The OGC WPS DescribeProcess request indicates a detailed description of the process with a specification for the required inputs and output parameters. The input of a WPS process has some of the following properties.

Property	Description
ows:Identifier	The unique identifier for the input.
ows:Title	The name of the input.
@minOccurs	It specifies the minimum required occurrence of the input. When the value of the @minOccurs is 0, this implies that the input is optional. On the other hand, if the value of @minOccurs is 1, the input is mandatory.
ows:Abstract	The description of the input
ows:DataType	The data type of the input object. However, it is mostly used when the input is not a geodata, for instance, numeric, Boolean or textual inputs. In case of a vector or raster input, the data type is obtained from the mime type format. If the mime type contains an XML or GML or JSON or WKT, then the data type is treated as a vector or geom. Otherwise, if the mime type contains an image, then the data type is treated as coverage or raster.

When we consider QGIS workflow interchange format, the input has the following properties.

Property	Description
identifier	The keyword for the identifier is assigned based on the name of the input feature specified by the user. Its value is a JSON object.
name	Name of the input
processing.core.parameters.ParameterVector processing.core.parameters.ParameterRaster	This can be a vector or raster. It was highlighted in Section 3.4 that QGIS workflow interchange format does not store metadata for textual inputs.

optional	Optional indicating whether an input is mandatory or not. A mandatory input has a Boolean value of true.
description	Description of the input.
value	Value of the input.

After considering the schema for the input of the WPS process, ILWIS operations, and QGIS algorithms, this research came up with the following input properties for the standard workflow interchange format.

A JSON schema for the input property was produced as illustrated in Listing 5.4.

Property	Description
id	Integer data type referring to the index of the input in the inputs array object.
identifier	String data type for the identifier
name	String data type
type	This represents the data type of the input item. The inputs were categorized into the following data types; Coverage, Vectors, Boolean, Text and Numeric.
description	String data type.
optional	Boolean data type.
URL	URL specifying the path to input data. This is used when data is passed by reference
value	The value of the input in case data is passed by value.

Listing 5.4: JSON schema for an operation's input

1.	{
2.	"\$schema": "http://json-schema.org/draft-04/schema#",
3.	"type": "object",
4.	"properties": {
5.	"id": {
6.	"type": "integer"
7.	},
8.	"identifier": {
9.	"type": "string"
10.	},
11.	"name": {
12.	"type": "string"
13.	},
14.	"type": {
15.	"type": "string"
16.	},
17.	"description": {
18.	"type": "string"
19.	},
20.	"optional": {
21.	"type": "boolean"
22.	},

```

23.         "url": {
24.             "type": "string"
25.         },
26.         "value": {
27.             "type": "string"
28.         }
29.     },
30.     "required": [
31.         "id",
32.         "identifier",
33.         "name",
34.         "type",
35.         "description",
36.         "optional",
37.         "url",
38.         "value"
39.     ]
40. }

```

d.) Outputs

ILWIS workflow modeler and WPS DescribeProcess requests have the same schema for a process output as the input. However, QGIS has a different output schema from its inputs. It only contains the description of the output and the screen coordinates of the output. It was mentioned in Section 3.3.2 that since QGIS does not store a detailed definition for its output, conversion of QGIS workflow to an independent workflow format can be a bottleneck.

This is because very little information can be deduced from their output schema. This research, therefore, recommends using the schema definitions used by ILWIS and WPS process description. The following output properties are suggested for the standard workflow interchange format. A corresponding JSON schema for the output object was produced as illustrated in

Listing 5.5.

Property	Description
id	An integer data type representing the index of the output in the outputs array object.
identifier	String data type.
name	String data type.
type	This represents the data type of the input item. The inputs were categorized into the following data types; Coverage, Vectors, Boolean, Text and Numeric.
description	String data type.
value	The raw value of the input.

Listing 5.5: JSON schema for an operation's output

```

1. {
2.     "$schema": "http://json-schema.org/draft-04/schema#",

```

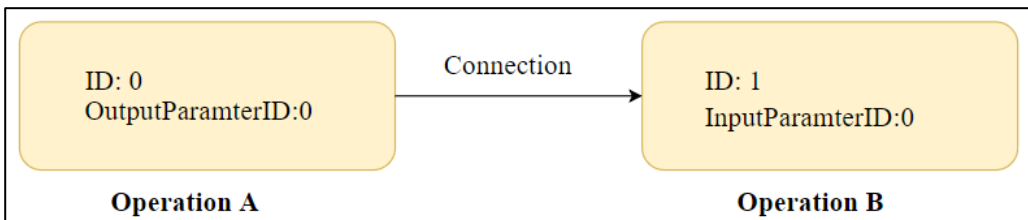
```

3.  "type": "object",
4.  "properties": {
5.    "id": {
6.      "type": "integer"
7.    },
8.    "identifier": {
9.      "type": "string"
10.   },
11.    "name": {
12.      "type": "string"
13.    },
14.    "value": {
15.      "type": "string"
16.    },
17.    "description": {
18.      "type": "string"
19.    },
20.    "type": {
21.      "type": "string"
22.    }
23.  },
24.  "required": [
25.    "id",
26.    "identifier",
27.    "name",
28.    "value",
29.    "description",
30.    "type"
31.  ]
32.  }

```

D. Connections

This object represents the linking of operations. An output of operation becomes an input to another operation. BPMN, OGC GPW, ILWIS, and QGIS address the issues of process chaining differently. Whereas WPS uses nested XML for chaining processes, ILWIS uses connection parameters of processes and inputs. BPMN, on the other hand, uses *incoming* and *outgoing* elements to specify data flow. Section 2.2 illustrates the concept of building a topological relationship between processes from a directed acyclic graph (DAG) model. Given two operations A and B with process IDs 0 and 1, this concept can be illustrated in the following two notations.



i. Using the operation IDs

The connection is from *OperationID 0* → *OperationID 1*

ii. Using the parameter IDs

The data flow is from *OutputParameterID 0* → *InputParameterID 0*

The recommended properties of the connections are explained in the following manner. A JSON schema conforming to these properties for the connection object was produced as illustrated in Listing 5.6.

Property	Description
fromOperationID	This is an integer value which represents the ID of the parent operation from which the connection originates.
toOperationID	This is an integer value which represents the ID of the child operation to which the connection is made.
fromParameterID	This is an integer value representing the ID of the output data of the parent operation.
toParameterID	This is an integer value representing the ID of the input data of the child operation.

Listing 5.6: JSON Schema for connection property

```

1. {
2.   "$schema": "http://json-schema.org/draft-04/schema#",
3.   "type": "object",
4.   "properties": {
5.     "fromOperationID": {
6.       "type": "integer"
7.     },
8.     "toOperationID": {
9.       "type": "integer"
10.    },
11.    "fromParameterID": {
12.      "type": "integer"
13.    },
14.    "toParameterID": {
15.      "type": "integer"
16.    }
17.  },
18.  "required": [
19.    "fromOperationID",
20.    "toOperationID",
21.    "fromParameterID",
22.    "toParameterID"
23.  ]
24. }
```

When the individual schemas for the workflow elements described in the previous sections are combined, we develop a standard JSON interchange schema which can be adopted by software developers to represent their workflows. The complete JSON schema is shown in Appendix A. while the visual presentation is as illustrated in the class diagram of Figure 5.2. A workflow engine that is implemented in Chapter 6 maps the constructs of this standard interchange schema to the workflow interchange formats of specific software packages that allow transformation of workflows from one WfMS to another. This

enables sharing of workflows across different software packages without the need to recreate the workflows in a different environment.

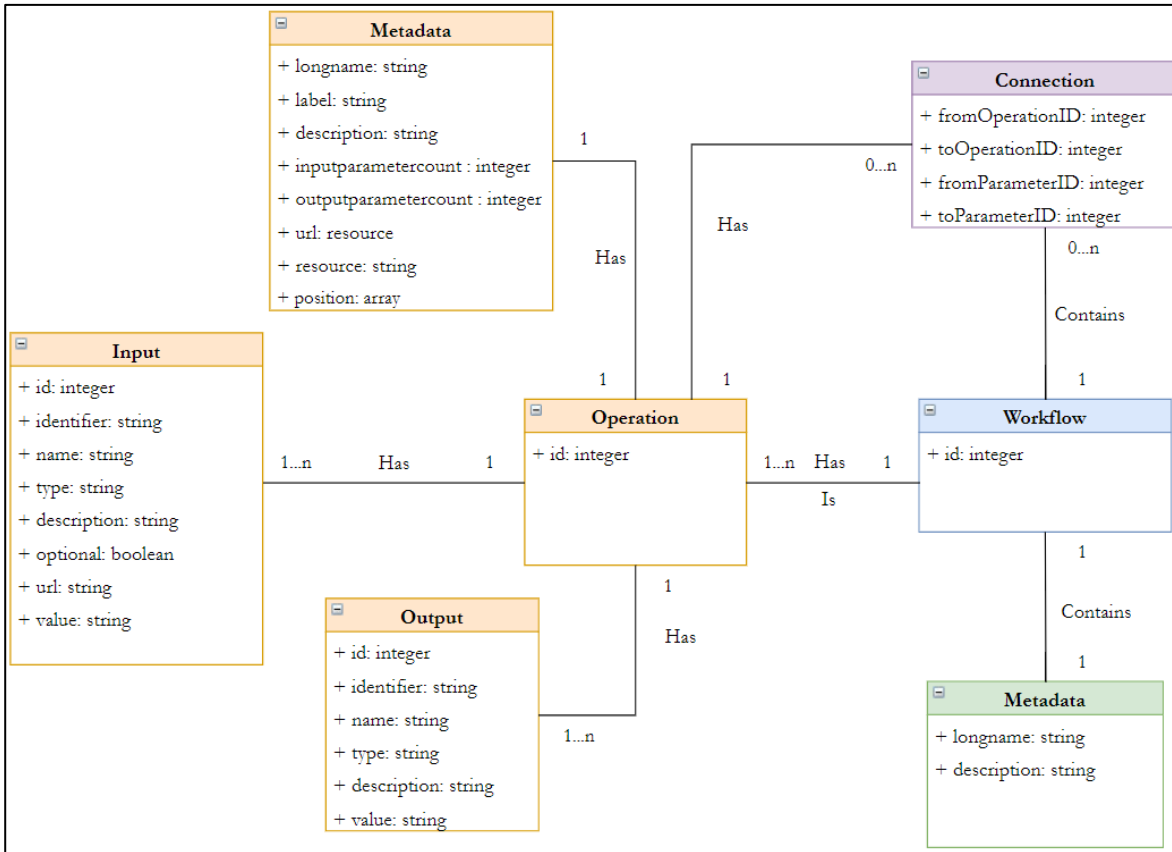


Figure 5.2: Class diagram for the Workflow Schema

5.2. Provenance Support for Reproducibility

Reproducibility allows a workflow created for a particular scientific problem to be reused by different users through repetition of steps to produce scientifically similar results. In Section 2.3.1, we discussed that to ensure reproducibility and repeatability, sufficient provenance information is desired. Provenance is used with workflows to capture information such as processes and their execution environment, input parameters provided to processes, a log of processes, connections, intermediary and final outputs. We discussed four main factors affecting reproducibility in Section 2.4 which includes the availability of third-party resources, nature of the input data, execution environment and provision of enough metadata for the workflow. The standardized workflow interchange schema mentioned in the previous section captures provenance for most of the required information to ensure successful reproducibility. For instance, it stores information about the third-party resources such as web services which include the name of the resource, the URL of the service provider, the internal name of the process, input and output parameters with their definitions and the connections between the processes. The OGC standards that were considered in Chapter 4 provide a framework for defining how data is shared among the processes. Since these standards are stable and are not prone to change frequently, they make it possible to reuse data hosted in remote databases.

Provenance information about the resources makes it possible to discover processes and data required to reproduce workflows in different WfMSs. This involves the detection of corresponding operators, for instance, internal names of processes in different WfMSs making it easy to reconstruct the workflow based on the input and output requirements of the target WfMS's operation. One method of discovering processes and data from provenance information in a workflow is by using semantic web technologies. This has been demonstrated by (Ubels, 2018). Due to time constraints, this research does not implement his method. However, we implement a simple search alternative to illustrate reproducibility of geoprocessing functions in different GIS WfMSs from a list of selected operations illustrated in Appendix D. The flowchart illustrated in Figure 5.3 describes the steps required to find the best match for a process name of a different GIS tool by a specified keyword. The first step in the flowchart uses keywords comprising of label, longname and description of the process obtained from the provenance of the workflow in the JSON interchange format to search the database for processes with similar keywords. The search result in an array of matching processes and the number of hits from the keyword. The next step determines if the array is empty in which case the process terminates implying there was no search result. In case the size of the array is greater than zero, the next step is invoked where the processes are listed ordered by the number of hits found. The process with the highest number of hits is chosen as the best match. The algorithm represented in the flowchart is implemented in 6.6.

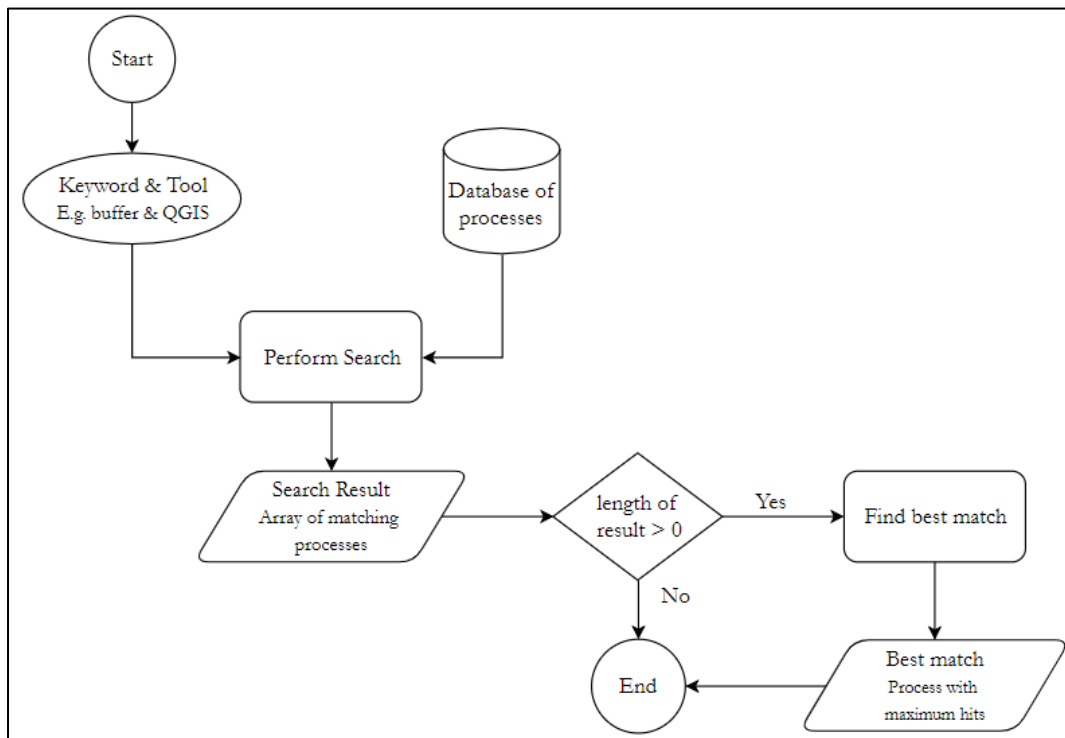


Figure 5.3: Flowchart for process discovery

5.3. REST API to Support Reuse

In this section, we provide a means in which software developers can reuse our services to support sharing and reproducibility of workflows. A RESTful API was implemented with the endpoint definitions shown in Table 5.2 capable of receiving and executing requests from users.

Table 5.2: RESTful API for Service Reuse

REST endpoint	Purpose
http://130.89.221.193:75/workflow/execute Body/Payload: Workflow (JSON text) Headers: {"content-type": "Application/json"}	Workflow execution
http://130.89.221.193:75/workflow/transform/ < string:source>/ < string:target> Body/Payload: Workflow (JSON text) Headers: {"content-type": "Application/json"} Or Body/Payload: Workflow (XML text for BPMN source) Headers: {"content-type": "text/xml"}	Workflow transformation

To execute a workflow, use the REST API, the workflow JSON format is loaded as a payload to the HTTP POST request. JSON specification for the workflow is based on the schema defined in the previous section.

To transform a workflow, the user must specify the *source* specification and the *target* in the endpoint. For instance, the following are valid examples of REST endpoints. The workflow specification of the source system is assigned to the payload in the POST request.

REST endpoint¹¹	Transformation
http://130.89.221.193:75/workflow/transform/pim/bpmn	Platform independent workflow specification to BPMN specification.
http://130.89.221.193:75/workflow/transform/pim/qgis	Platform independent workflow specification to QGIS specification.
http://130.89.221.193:75/workflow/transform/ilwis/qgis	ILWIS to QGIS workflow specification.
http://130.89.221.193:75/workflow/transform/qgis/bpmn	QGIS to BPMN workflow specification
http://130.89.221.193:75/workflow/transform/pim/ilwis	Platform independent workflow specification to ILWIS specification.

¹¹ **Disclaimer:** The links provided in the examples may have been changed by the service providers.

6. PROTOTYPE IMPLEMENTATION

In the previous chapters, we discussed two methods which can facilitate sharing and reproduction of geoprocessing workflows. These included shareable processes which are exposed as web services in composing workflows and also providing a standard workflow interchange format for representing workflows. We discussed in Chapter 3 that to compose workflows from web services, we require a web-based workflow client and a backend workflow engine which provides functionalities which current WfMSs cannot offer. The most important feature which current GIS WfMSs cannot provide is the ability to use cloud-based processing services. Cloud-based computing allows users to perform geocomputation utilizing state-of-the-art high-performance computing technologies which cannot be provided by personal computers. In Chapter 4, we discussed how we can use web services to compose workflows using the standards proposed by OGC for web processing services and data services. We also discussed the OGC process chaining which provides another way of modelling the sequence of workflow execution. The fifth chapter of this research proposed a standard workflow interchange format based on a JSON schema which can be used to create a platform independent model for transforming workflows from different WfMSs. We found out several constructs of particular WfMS which are semantically related and can be mapped using the standard interchange format.

In this chapter, we take a different approach which involves the development of a prototype system that can achieve all the concepts we discussed in the previous chapters. In the next Section, we present the architecture of the prototype system and its requirements.

6.1. System Architecture

A system architecture consists of the components making up a system, their functions, and interactions to provide the desired objective of the system. The system's components follow a design principle in computer science known as *separation of concerns* where each of them addresses a separate concern (Singh, 2016). We used this principle in our implementation to ensure a scalable application that supports reuse of modules and also provide independent developments and maintenance. Figure 6.1 illustrates this principle through the multi-tier client-server architecture which separates the presentation layer, processing layer and database layer. The presentation layer is visible to users and provides a thin client allowing them to create and modify workflows from available web services. The processing layer is made up of distributed processing servers providing OGC WPS compliant and non-OGC compliant RESTful services.

Abstracted from the users is the processing engines from which these web processing services lie which include ILWIS, QGIS, 52North WPS Servers, etc. The database layer is responsible for providing access to data through the data services such as WFS, WCS and SWE's SOS. For simplicity, the system's components were distinguished into two categories: client-side components and the server-side components.

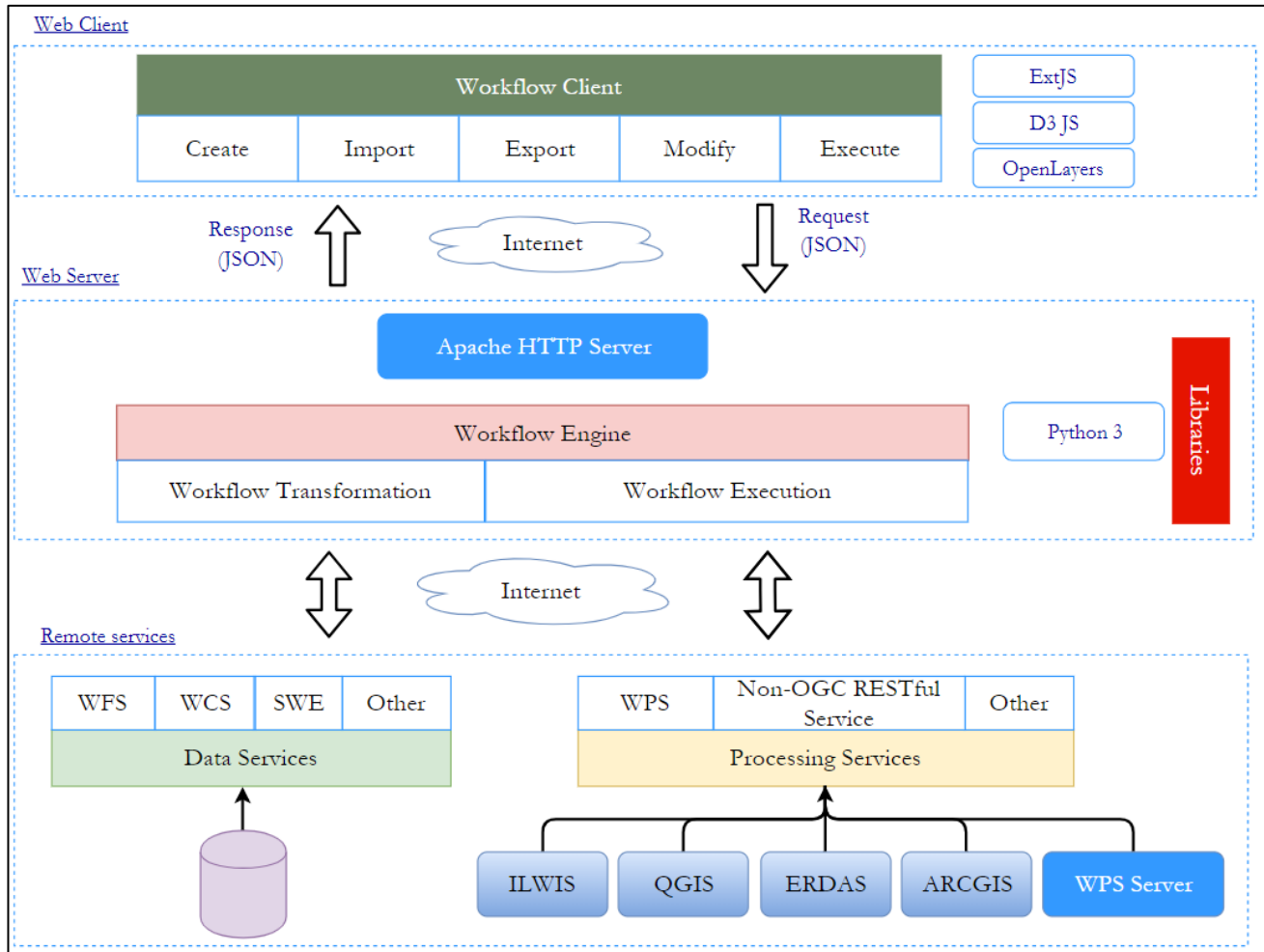


Figure 6.1: System Architecture

6.1.1. Client-Side Components

The client-side components include the software and libraries which are needed to build the client application. Since this is a web application, it requires a JavaScript enabled web browser. Most of the modern web browsers are JavaScript enabled and can be used to run the web application. This particular web application was built using the following JavaScript libraries.

Ext JS

Ext JS is a popular JavaScript framework for building interactive cross-platform web applications. Ext JS uses scripting techniques of AJAX, DHTML, and DOM which are essential for asynchronous programming and it complies with the model-view-controller (MVC) architectural pattern thereby granting separation of concerns. Making a graphical user interface (GUI) with Ext JS does not involve a lot of programming work since most of the components are inherited from already implemented classes. We used ExtJS to build the GUI of the applications and provide data access using AJAX technology.

D3 JS

D3 JS is a JavaScript library used in creating dynamic and interactive data visualizations in web browsers. It uses the client stack web technologies mainly Scalable Vector Graphics (SVG), HyperText Markup Language (HTML) and Cascading Style Sheet (CSS). We use the D3 library in our prototype system for visualization of the workflow elements using BPMN diagrams. The visual workflow was translated automatically to a textual JSON representation using the standardized workflow interchange format proposed in Chapter 5. We also used D3 JS to display time-series of sensor data obtained using the Sensor Observation Service (SOS).

OpenLayers

OpenLayers is another JavaScript library used for displaying map data in the web browsers. OpenLayers offers a dynamic display of maps and allows users to interact with the features in the map using map events. OpenLayers provides a display for map tiles and vector data using WMS and WFS respectively. We used OpenLayers to display input and output maps of the workflow.

6.1.2. Server-Side Components

The server-side components of the web application provide the functions of a workflow engine as well as support interactions with the database. Some of the functions include the transformation of workflow from one interchange format to another, the orchestration of services to obtain the execution sequence and coordinate the execution of the workflow. For demonstration purpose, we implemented non-OGC RESTful services for ILWIS operations and other RESTful services to enable execution of workflow using other REST-based clients. The following components are part of the server.

Apache HTTP Server

Apache HTTP server was used to render web pages to the client as well as provide an interface between the client and the other server applications. The version of Apache HTTP server used for this demonstration was 2.4. The installation and setup of Apache HTTP Server are available as documentation in the Apache website.

Apache Tomcat and GeoServer

We installed GeoServer in two servers to act as providers for distributed processing and data services. Using the GeoServer manual, we set up the WCS, WFS, and WPS. GeoServer is a Java Servlet application, and that was the motivation for using Apache Tomcat since it is built to run Java servlet applications. This set up used Apache Tomcat version 9 and GeoServer version 2.8. Since these services run on Java, it required the installation of Java Runtime Environment (JRE) version 8. This research also made use of the GeoServer managed by 52° North organization to offer data and processing services.

Python

The prototype system used Python as a server-side programming language. The choice for python was inspired by the fact that it has a lot of libraries developed for geocomputation such as GDAL and Numpy. A lot of GIS software also have a python module which provides an API for which this prototype can consume their functions. For instance, ILWIS and QGIS python connectors. The version of Python programming used was Python 3.6.

GIS Processors

To enable us to study and demonstrate the functions of our system regarding sharing processes and workflows, this research relied on ILWIS and QGIS. These are open source GIS software applications which gave us the opportunity to explore their internal operations using their python connectors.

6.2. Generic Workflow Client

As a proof of concept, a generic workflow client was developed (see Figure 6.2) which allows the visual composition of workflow from web services. The workflow client has three main panels; the first panel is made of up of web services which are further divided into processing and data services sub-panels. The processing services are listed in a tree view with the root referring to the name of the processing server offering the services while the children are made of the individual processes. The default set up is made up of five processing servers offering OGC compliant WPS and non-OGC compliant RESTful processing services. The metadata for a particular processing service can be viewed by right-clicking the process from the tree and selecting the appropriate menu from the menu item. The data services sub-panel provide a list of WCS, WFS and SOS web services that are used for this demonstration. The second panel comprises of a visual editor for composing workflows from the listed web services. The processing services are added to the editor by dragging and dropping the listed services from the tree view which automatically draw a rectangular object for the chosen process. Whenever a new web service is added to the editor panel, a textual representation of the workflow as a JSON object is automatically created from the visual representation based on the schema which was discussed in Section 5.1.2. The JSON text similar to the one shown in Listing 6.1 can be saved to a local folder and reused in the future to run the same workflow with different data using the same processes. The third panel is responsible for the visualization of data and results in a map and chart.

To achieve structural composability, we use nodes to represent processes and edges to define connections between processes. The user specifies the sequence of processing by creating links from the source to the target process. Swapping processing services of different service providers is possible to enable users to perform the same operation in a separate processing server from the former and achieve high reliability. The data services are passed to the processes by reference to the path of the data. This can be

accomplished by manually typing the path of the data in the settings of the process or by dragging and dropping the service from the list of data services available. The workflow client takes care of the static syntactic compatibility by checking that the data type of a source process output is the same for the target process input. In case of different data types, for example, connecting a raster output to a vector input, an error message is displayed to warn the user. Verifying the workflow for semantic composability discussed by Diniz (2016) is not supported for this implementation because of the different schema used for our JSON representations of the workflow.

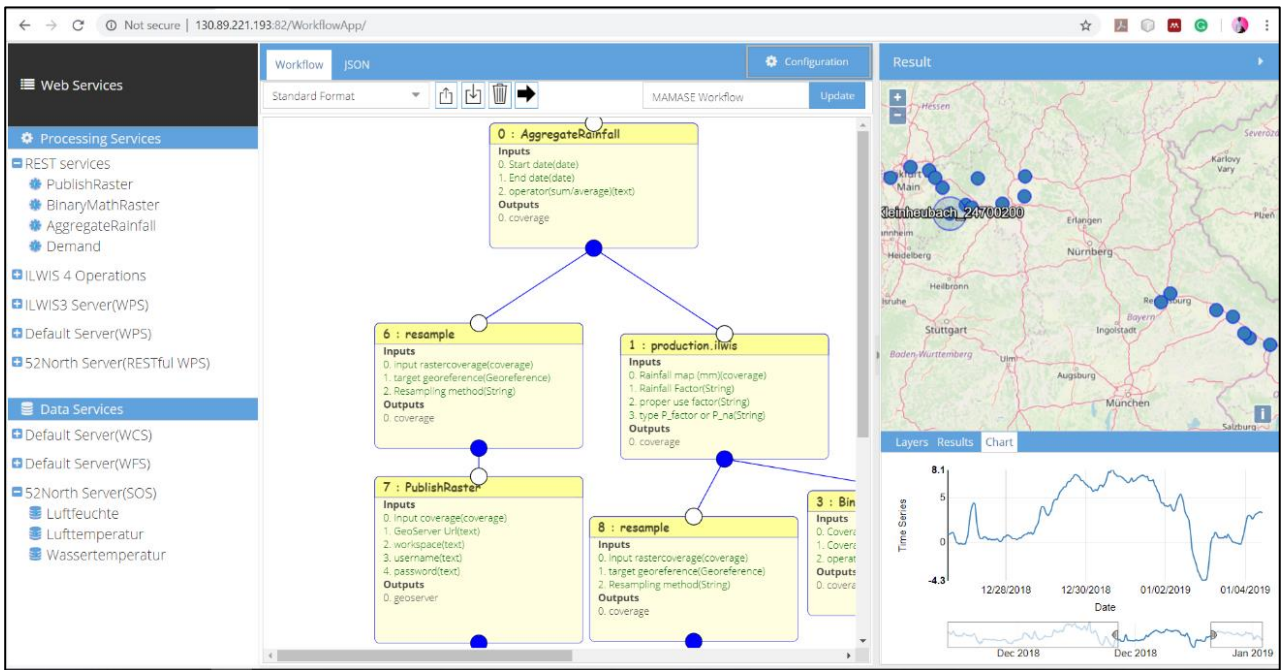
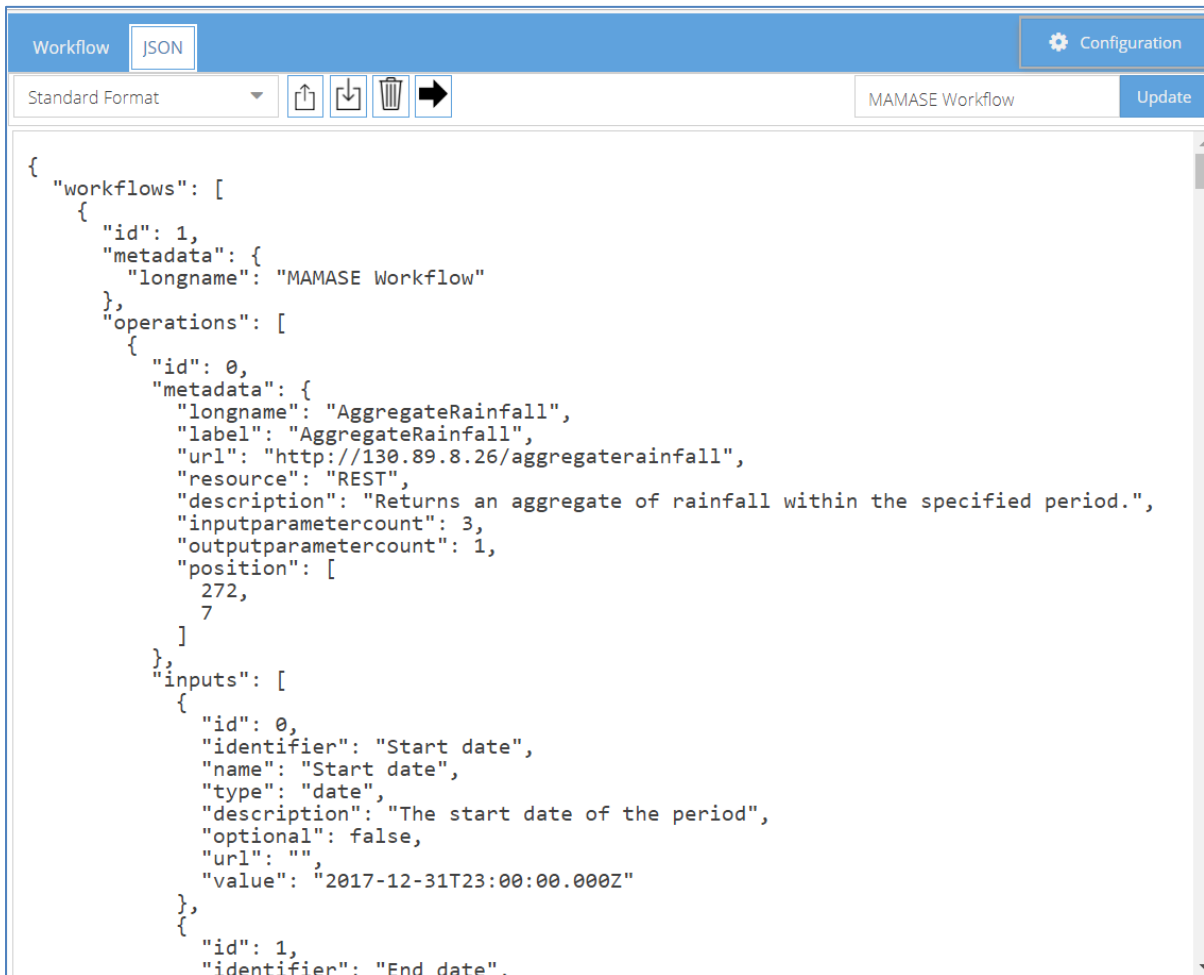


Figure 6.2: The Generic Workflow Client's User Interface

This interface was built using the ExtJS framework and the map rendered using OpenLayers which are already discussed in the previous Section. The source codes for this implementation are available in the GitHub¹².

¹² <https://github.com/robertohuru/WorkflowApp>

Listing 6.1: Snippet of the JSON Representation of a Workflow



```
{
  "workflows": [
    {
      "id": 1,
      "metadata": {
        "longname": "MAMASE Workflow"
      },
      "operations": [
        {
          "id": 0,
          "metadata": {
            "longname": "AggregateRainfall",
            "label": "AggregateRainfall",
            "url": "http://130.89.8.26/aggregaterainfall",
            "resource": "REST",
            "description": "Returns an aggregate of rainfall within the specified period.",
            "inputparametercount": 3,
            "outputparametercount": 1,
            "position": [
              272,
              7
            ]
          },
          "inputs": [
            {
              "id": 0,
              "identifier": "Start date",
              "name": "Start date",
              "type": "date",
              "description": "The start date of the period",
              "optional": false,
              "url": "",
              "value": "2017-12-31T23:00:00.000Z"
            },
            {
              "id": 1,
              "identifier": "End date",

```

To run the workflow, the JSON file is sent via HTTP POST to the workflow engine for execution. The workflow client also allows users to download the resulting data of each operation using WCS and WFS. The supported WCS output format is Geotiff while WFS is GeoJSON.

6.3. Data Services

Our implementation makes use of the OGC standards for data access and sharing for raster, vector and sensor data. These standards enable users to create, share and combine the traditional satellite and in-situ data with the crowdsourced geoinformation or Volunteered Geographic Information. Combination of data obtained from the three sources has several benefits when used in a workflow. One of the most important benefit is the ability to incorporate the use of most recent data in a workflow which can supplement traditional satellite and in-situ data. This becomes useful for scientists involved in research including disaster management, air pollution, water resource monitoring and management among others. The OGC specified three standards for data services which we already discussed in Section 4.2. In the following section, we discuss the implementation of these standards in our prototype system. For the

demonstration of these services, we used GeoServer. However, the same functions can be provided using other servers such as the Mapserver.

6.3.1. Web Feature Service

In Section 4.2.1, we discussed three operations for the OGC Web Feature Service (WFS) that we found to be relevant for this research. These operations were the GetCapabilities, DescribeFeatureType, and GetFeature. We implement GetCapabilities to retrieve the metadata of all the features within a given WFS server. The GetCapabilities operation requires the URL of the WFS server, service type, request and the version of the GeoServer. In the web client, users can specify the URL for their WFS server using the configuration section.

The response from a GetCapabilities is an XML text which we convert to a JSON format. From this response, we select five properties of the feature which include the feature's name, title, abstract, default coordinate system and the WFS GeoJSON path for retrieving the data. We implement DescribeFeatureType to help us get more information about a particular feature. For our proof of concept, we preferred using GeoJSON because it is lightweight and integrates very fast in modern browsers and with OpenLayers. We, however, note that some WFS servers provide their data in other formats which are not GeoJSON.

Table 6.1: WFS Operations

WFS GetCapabilities	http://130.89.221.193:85/geoserver/ows? service=WFS& request=GetCapabilities& version=1.0.0
DescribedFeatureType	http://130.89.221.193:85/geoserver/wfs? request=DescribeFeatureType& version=1.0.0& TypeName= group1:waterbodies
GetFeature	http://130.89.221.193:85/geoserver/wfs? request=GetFeature& version=1.0.0& TypeName=group1:waterbodies& Outputformat=application/json

Listing 6.2: Snippet for the Python implementation of WFS GetCapabilities

```

1. #!C:/Users/Bob/AppData/Local/Programs/Python/Python36/python
2. import json
3. import cgi
4. import requests
5. import xmltodict
6.

```

```

7. print("Content-type: application/json")
8. print()
9.
10.     params = cgi.FieldStorage()
11.     # URL of the WFS Server
12.     url = params.getvalue('url')
13.     if url is None:
14.         url = "http://130.89.8.26:85/geoserver/ows?"
15.
16.     # Result of the GetCapabilities
17.     results = requests.get(url+"service=WFS&request=GetCapabilities")
18.     features = []
19.     if results.text == "":
20.         features = []
21.     else:
22.         # parse the XML response to a JSON object
23.         jsonResponse = xmltodict.parse(results.text)
24.         for row in jsonResponse['wfs:WFS_Capabilities']['FeatureTypeList']['FeatureType']:
25.             feature = {}
26.             if "mapserv.exe?" in url:
27.                 feature['url'] = url + "service=WFS&request=GetFeature&typeName="+row['Name']+"&outputFormat=geojson&srsname=EPSG:3857"
28.             else:
29.                 feature['url'] = url + "service=WFS&request=GetFeature&typeName=" + row['Name'] + "&outputFormat=application/json"
30.                 feature['name'] = row['Name']
31.                 feature['title'] = row['Title']
32.                 feature['abstract'] = row['Abstract']
33.                 feature['defaultCRS'] = row['DefaultCRS']
34.                 results = requests.post(url + "service=WFS&request=DescribeFeatureType&typeName="+row['Name']+"&outputFormat=application/json")
35.                 results = json.loads(results.text)
36.                 feature['properties'] = results['featureTypes']
37.                 features.append(feature)
38.
39.     print('{"success": "true", "features":', json.dumps(features),
        '}')

```

We implemented the code snippet in Listing 6.2 to retrieve metadata information for features using the WFS GetCapabilities and DescribeFeatureType operations. The result was wrapped in a JSON object (Listing Listing 6.3) which is submitted to the workflow client. The workflow client implements an ExtJS tree view panel where the leaf represents the title for each feature.

Listing 6.3: JSON object for GetCapabilities request

```

3  "features": [
4  {
5      "title": "DMintake_HERB_kg_cons_nrdays",
6      "url": "http://130.89.8.26:85/geoserver/ows?service=WFS&request=
           =GetFeature&typeName=maris_mamase
           :DMintake_HERB_kg_cons_nrdays&outputFormat=application/json",
7      "properties": [
8      {
9          "typeName": "DMintake_HERB_kg_cons_nrdays",
10         "properties": [
11         {
12             "type": "gml:MultiPolygon",
13             "maxOccurs": 1,
14             "nillable": true,
15             "name": "the_geom",
16             "localType": "MultiPolygon",
17             "minOccurs": 0
18         },
19         {
20             "type": "xsd:number",
21             "maxOccurs": 1,
22             "nillable": true,
23             "name": "feature_va",
24             "localType": "number",
25             "minOccurs": 0
26         }
27         ]
28     }
29 ],
30 "name": "maris_mamase:DMintake_HERB_kg_cons_nrdays",
31 "abstract": null,
32 "defaultCRS": "urn:ogc:def:crs:EPSG::21036"
33 },

```

6.3.2. Web Coverage Service

The OGC Web Coverage Service discussed in Section 4.2.2 specifies three operations which we considered relevant for this research. These operations include the GetCapabilities, DescribeCoverage, and GetCoverage. The implementation for WCS is similar to that of WFS which has been discussed in the previous Section. However, instead of GeoJSON, we now use Geotiff as the data format. Our choice for using Geotiff for our proof of concept is motivated by the fact that most GIS software can read GeoTIFF files and thus making this file format platform independent.

Table 6.2: WCS Operations

GetCapabilities	http://130.89.8.26:85/geoserver/ows? service=WCS& request=GetCapabilities
DescribeCoverage	http://130.89.8.26:85/geoserver/ows? service=WCS& request=DescribeCoverage& coverageid=maris_mamase:carcap_kg_23m& version=1.0.0
GetCoverage	http://130.89.8.26:85/geoserver/ows?version=2.0.0&

	service=WCS& request=GetCoverage& coverageid=maris_mamase:DMintake_kg_23m_nrdays& format=image/geotiff
--	---

6.3.3. Sensor Observation Service

The Sensor Observation Service discussed in Section 4.2.3 specifies three operations which we considered relevant for this research. These include the GetCapabilities, DescribeSensor, and GetObservation. We implemented GetCapabilities and Describe Sensor to obtain metadata information about sensors and their observed properties.

Table 6.3: SOS Operations

GetCapabilities	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=GetCapabilities& service=SOS
DescribeSensor	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=DescribeSensor& service=SOS& procedure=Lufttemperatur-Frankfurt_Osthafen_24700404& outputformat=text/xml;subtype="sensorml/1.0.1"& version=1.0.0
GetObservation	https://pegelonline.wsv.de/webservices/gis/gdi-sos? request=GetObservation& service=SOS& procedure=Lufttemperatur-Frankfurt_Osthafen_24700404& version=1.0.0& offering=LUFTTEMPERATUR& observedProperty=Lufttemperatur& featureOfInterest=Frankfurt_Osthafen_24700404& responseformat=text/xml;subtype="om/1.0.0"

6.4. Processing Services

The processing services are vital for composing workflows in our web client. Without a process, we cannot create a workflow to consume available data. Our implementation gives users the opportunity to add the URL or endpoints to the processing services which they would want to use in defining their

workflows. To achieve this, we observe two approaches where one is based on a standard OGC WPS, and the other is based on Non-OCG RESTful services.

6.4.1. OGC Web Processing Service

In Section 4.3.1, we discussed the three operations of the OGC Web Processing Service which are relevant for this research. They include the GetCapabilities, DescribeProcess and Execute operations. We implement the GetCapabilities and DescribeProcess to help retrieve metadata information about web processes. A Python function was implemented which uses the HTTP GET method to obtain this information from a WPS server specified by the user. An OGC compliant WPS was used which was provided by the implementation of GeoServer WPS extension ¹³ and 52North WPS solution ¹⁴. The response of the GetCapabilities and DescribeProcess operations are used to build a list of operations which are then sent to the workflow client in a JSON object.

Table 6.4: WPS Operations

GetCapabilities	http://130.89.221.193:85/geoserver/ows? service=WPS& request=GetCapabilities
DescribeProcess	http://130.89.221.193:85/geoserver/ows? service=WPS& request=DescribeProcess& identifier=gs:Centroid

During our discussion of the standard schema for sharing workflows in Section 5.1.2, we proposed that the schema for a processing service should have four main attributes. These include the id, metadata, inputs, and outputs. Based on the attributes of OGC WPS discussed in Section 3.1.2 and the mapping between the OGC WPS XML schema and our proposed standard schema as was observed in Section 5.2.3, we implemented a method to automatically obtain a JSON object with a list of all the processes in a WPS server. This method makes use of the WPS GetCapabilities and DescribeProcess operations. For instance, the GetCapabilities and DescribeProcess operations in the table above returns the XML responses whose snippet are as follows.

Listing 6.4: OGC WPS GetCapabilities response for gs:Centroid operation

```

1. </wps:Process>
2. <wps:Process wps:processVersion="1.0.0">
3. <ows:Identifier>JTS:centroid</ows:Identifier>
4. <ows:Title>Centroid</ows:Title>
5. <ows:Abstract>
6. Returns the geometric centroid of a geometry. Output is a single po
   int. The centroid point may be located outside the geometry.
7. </ows:Abstract>

```

¹³ <https://docs.geoserver.org/stable/en/user/services/wps/index.html>

¹⁴ <http://geoprocessing.demo.52north.org:8080/latest-wps/WebProcessingService>

```
8. </wps:Process>
```

Listing 6.5: XML Snippet for OGC WPS DescribeProcess for gs:Centroid

```
1. <wps:ProcessDescriptions xmlns:xs="http://www.w3.org/2001/XMLSchema
   " xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:wps="http://www.
   opengis.net/wps/1.0.0" xmlns:xlink="http://www.w3.org/1999/xlink" x
   mlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xml:lang="en" service="WPS" version="1.0.0" xsi:schemaLoc
   ation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/
   wps/1.0.0/wpsAll.xsd">
2. <ProcessDescription wps:processVersion="1.0.0" statusSupported="tru
   e" storeSupported="true">
3. <ows:Identifier>gs:Centroid</ows:Identifier>
4. <ows:Title>Centroid</ows:Title>
5. <ows:Abstract>Computes the geometric centroids of features</ows:Abs
   tract>
6. <DataInputs>
7. <Input maxOccurs="1" minOccurs="1">...</Input>
8. </DataInputs>
9. <ProcessOutputs>...</ProcessOutputs>
10. </ProcessDescription>
11. </wps:ProcessDescriptions>
```

Listing 6.6: Code snippet for mapping of WPS process definition to standard JSON schema

```
1.
2. if url is None:
3.     url = "http://130.89.221.193:85/geoserver/ows?"
4. results = requests.get(url + "service=WPS&request=GetCapabilities")
5. if results.text == "":
6.     results = []
7. xpars = xmltodict.parse(results.text)
8. jsonjson1 = json.dumps(xpars)
9. d = json.loads(json1)
10.     processes = []
11.     for row in d['wps:Capabilities']['wps:ProcessOfferings']['wps
       :Process']:
12.         process = {}
13.         identifier = row['ows:Identifier']
14.         process['id'] = identifier
15.         # Add metadata to process
16.         metadata = {}
17.         metadata['resource'] = 'WPS'
18.         metadata['url'] = url
19.         metadata['description'] = abstract
20.         process['metadata'] = metadata
21.         response = requests.get(url + "service=WPS&request=Descri
       beProcess&identifier=" + identifier)
22.         response = xmltodict.parse(response.text)
23.         jsonjson2 = json.dumps(response)
24.         b = json.loads(json2)
25.
26.         if 'ProcessDescription' in b['wps:ProcessDescriptions']:
27.             for item in inputs:
28.                 input = {}
29.                 input['id'] = 0
```

```

29.         input['identifier'] = item['ows:Identifier']
30.         input['name'] = item['ows:Title']
31.         input['url'] = ""
32.         input['value'] = ""
33.         if item['@minOccurs'] == '0':
34.             input['optional'] = True
35.         else:
36.             input['optional'] = False

```

Using our implementation, we can obtain the corresponding JSON definition of the above responses based on our proposed standard schema as shown in the Listing below. The *id* corresponds to the identifier of the WPS operation. This particular operation has only one input which is a vector and one output which is also a vector. For our implementation, the vector data are assigned a data type named *geom* while the raster data type is assigned *coverage*. The metadata of the operation provides it a description, label, longname, URL of the WPS server where the operation resides, resource specifying that this operation is being offered as a WPS.

Listing 6.7: JSON representation for WPS gs:Centroid operation

```

415 {
416     "id": "gs:Centroid",
417     "inputs": [
418         {
419             "id": 0,
420             "url": "",
421             "description": "Input feature collection",
422             "value": "",
423             "optional": false,
424             "name": "features",
425             "identifier": "features",
426             "type": "geom"
427         }
428     ],
429     "metadata": {
430         "description": "Computes the geometric centroids of features",
431         "longname": "Centroid",
432         "inputparametercount": 1,
433         "outputparametercount": 1,
434         "url": "http://130.89.221.193:85/geoserver/ows?",
435         "resource": "WPS",
436         "label": "gs:Centroid"
437     },
438     "outputs": [
439         {
440             "id": 0,
441             "description": "result",
442             "value": "",
443             "name": "result",
444             "identifier": "result",
445             "type": "geom"
446         }
447     ]
448 },
449 {

```

The JSON object listing the WPS operations is then passed to the ExtJS tree view panel as a data store which is used to make the tree view in the web client. The WPS execute operation allows execution of WPS processes by performing a POST request to the WPS server URL with a payload containing the XML definition of the process. Listing 6.8 shows an example of a WPS execute body in XML format. An implementation which automatically generates an executable XML script using WPS specifications is implemented in Section 6.4.1.

Listing 6.8: Sample WPS Execute Body.

```

1. <?xml version="1.0" encoding="UTF-
8"?><wps:Execute version="1.0.0" service="WPS" xmlns:xsi="http://ww
w.w3.org/2001/XMLSchema-
instance" xmlns="http://www.opengis.net/wps/1.0.0" xmlns:wfs="http:
//www.opengis.net/wfs" xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1" xmlns:gml="http://www.o
pengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="h
ttp://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://www.w3.org/199
9/xlink" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http:
//schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
2.   <ows:Identifier>gs:BufferFeatureCollection</ows:Identifier>
3.   <wps>DataInputs>
4.     <wps:Input>
5.       <ows:Identifier>CoverageA</ows:Identifier>
6.       <wps:Reference mimeType="image/tif" xlink:href="http://130.89
.8.26:85/geoserver/maris_mamase/ows?version=2.0.0&service=WCS&reque
st=GetCoverage&coverageId=maris_mamase:DMintake_kg_23m_nrdays&forma
t=image/geotiff" method="GET"/>
7.     </wps:Input>
8.     <wps:Input>
9.       <ows:Identifier>CoverageB</ows:Identifier>
10.      <wps:Reference mimeType="image/tif" xlink:href="http://
130.89.8.26:85/geoserver/maris_mamase/ows?version=2.0.0&service=WCS
&request=GetCoverage&coverageId=maris_mamase:DMprod_kg_ha_250m2&for
mat=image/geotiff" method="GET"/>
11.    </wps:Input>
12.    <wps:Input>
13.      <ows:Identifier>operator</ows:Identifier>
14.      <wps>Data>
15.        <wps:LiteralData>add</wps:LiteralData>
16.      </wps>Data>
17.    </wps:Input>
18.  </wps>DataInputs>
19.  <wps:ResponseForm>
20.    <wps:RawDataOutput mimeType="image/tif">
21.      <ows:Identifier>result</ows:Identifier>
22.    </wps:RawDataOutput>
23.  </wps:ResponseForm>
24. </wps:Execute>

```

6.4.2. Non-OGC Compliant RESTful Services

Non-OGC compliant RESTful services are not easy to model in a workflow because they don't follow any standards making it almost difficult to obtain their metadata information which is required to understand the service requirements concerning inputs and output parameters. In Section 4.3.2, we discussed how

common RESTful services like coordinate transformation could be interpreted and mapped into a similar WPS specification. We use the same concept to implement four RESTful services to demonstrate how non-OGC compliant RESTful services can be accommodated in a workflow.

Table 6.5: Non-OGC Compliant RESTful Services

Name	Description	Example usage
AggregateRainfall	Aggregate CHIRPS rainfall data for a given start and end period. Returns a raster image. It uses ILWIS processes.	http://130.89.8.26/aggregaterainfall/2018-01-01/2019-01-01/sum
BinaryMathraster	Returns a raster generated by pixel-by-pixel addition of two source rasters. Source rasters must have the same bounding box and resolution.	http://130.89.221.193:75/binarymathraster/path2raster1/pathr2aster2/add
PublishRaster	publishes a raster map to the specified GeoServer. It returns the namespace of the published map.	http://130.89.221.193:75/publish/raster/path2raster/GEOSERVERURL/workspace/username/password
Demand	Returns the biomass demand for the specified period. This REST service invokes an ILWIS workflow and passes the start and end dates.	http://130.89.8.26/demand/2018-01-01/2019-01-01

For an illustration of our implementation of the RESTful services, we only focus on the aggregaterainfall operation. The operation takes three inputs of data type string representing the start and end dates of the period of interest, and the type of aggregate operator to apply on the data. It returns a raster or a coverage. Therefore, we can implicitly define the JSON representation of this service using the following notation.

Listing 6.9: JSON representation for the AggregateRainfall RESTful service.

```

1 {
2   "id": 0,
3   "metadata": {
4     "longname": "AggregateRainfall",
5     "label": "AggregateRainfall",
6     "url": "http://130.89.8.26/aggregaterainfall",
7     "resource": "REST",
8     "description": "Returns an aggregate of rainfall within the specified period.",
9     "inputparametercount": 3,
10    "outputparametercount": 1
11  },
12  "inputs": [
13    {
14      "id": 0,
15      "identifier": "Start date",
16      "name": "Start date",
17      "type": "date",
18      "description": "The start date of the period",
19      "optional": false,
20      "url": "",
21      "value": ""
22    },
23    { },
24    { }
25  ],
26  "outputs": [ ]
27 }

```

The workflow client implementation allows users to define their REST endpoints and specify input and output requirements for the service. This automatically generates the JSON representation similar to the one used in the example above.

The screenshot shows a 'Processing Service' window with the following fields and values:

- Resource***: REST
- Name***: (empty)
- Enter URL***: http://130.89.221.193:75/binarymathraster
- Description***: (empty)
- Input Parameter Count**: 1
- Input 1***: Name (empty), Select data type... (dropdown)
- Output Parameter Count**: 1
- Output 1***: Name (empty), Select data type... (dropdown)
- Save**: button

Figure 6.3: RESTful Service Definition through the Workflow client

6.5. Workflow Engine

In Section 4.5, we discussed that our prototype would need a workflow engine component to coordinate execution of the workflow. We discussed that current GIS WfMSs have their workflow engines, but they

have a limitation on using web services. We also discussed that several BPMN compliant workflow engines are capable of executing service-based workflows using Business Process Execution Language (BPEL). However, we noted that most of them are commercial and require considerable human effort to compose and execute workflows. BPMN compliant workflow engines are also designed for business processes and do not support visualization of execution results. We, therefore, proposed to implement a workflow engine which offers a “one-stop shop” functionality for users to compose, execute and view results of their workflow composition just like it is possible in many GIS WfMSs. This section discusses the implementation of a workflow engine that can support the following functions which were discussed earlier.

- i. Translate the JSON representation of the workflow to an executable script which can be executed by the workflow engine. This involves automatic creation of WPS execute body using XML for OGC compliant WPS.
- ii. Control and coordinate the execution of the workflow by chaining web services according to the order of the service composition.
- iii. Generate downloadable results and provide users with the ability to view their result as a Web Mapping Service (WMS).
- iv. Transform workflow produced by one WfMS to another WfMS.

The implementation of this workflow engine used Python programming language and was built on PyCharm IDE.

6.5.1. Translating JSON Representation to Executable Script

The workflow engine can determine from the workflow definition if an operation belongs to OGC compliant WPS or non-OGC RESTful service. For the OGC WPS operation, we implement a method which is capable of generating the WPS Execute script using XML from a JSON representation. This method creates an executable WPS similar to the one discussed in Section 6.4.1 which then can be sent through an HTTP POST request to the WPS server URL with a payload containing the XML definition of the process. We use the function in Listing 6.10 to initialize the WPS header tag which specifies the OGC schema locations for WPS, WFS and WCS.

Listing 6.10: WPS Root element specification.

```

1. def wpsHead(self):
2.     root = Element('wps:Execute')
3.     root.set('service', 'WPS')
4.     root.set('version', '1.0')
5.     root.set('xmlns:xsi', 'http://www.w3.org/2001/XMLSchema-
    instance')
6.     root.set('xmlns', 'http://www.opengis.net/wps/1.0.0')
7.     root.set('xmlns:wfs', 'http://www.opengis.net/wfs')
8.     root.set('xmlns:wps', 'http://www.opengis.net/wps/1.0.0')
9.     root.set('xmlns:ows', 'http://www.opengis.net/ows/1.1')
10.     root.set('xmlns:gml', 'http://www.opengis.net/gml')
11.     root.set('xmlns:ogc', 'http://www.opengis.net/ogc')

```

```

12.         root.set('xmlns:wcs', 'http://www.opengis.net/wcs/1.1.1')
13.         root.set('xmlns:xlink', 'http://www.w3.org/1999/xlink')
14.         root.set('xsi:schemaLocation',
15.                 'http://www.opengis.net/wps/1.0.0 http://schemas
.opengis.net/wps/1.0.0/wpsAll.xsd')
16.         return root

```

Listing 6.11 shows a function which was implemented for creating an executable WPS script which is then sent to a WPS server through an HTTP POST request for execution. Line 3-44 is responsible for the creation of the XML body of the WPS while 48 is used to submit a request to the WPS server which executes the process based on the WPS definition.

Listing 6.11: Python Code Snippet for WPS Execute Implementation.

```

1. def executeWPS(operation, type='application/json'):
2.     # Create the Execute body of the Process
3.     root = WorkflowUtils.wpsHead(WorkflowUtils)
4.     label = operation['metadata']['label']
5.     ows_Identifier = SubElement(root, 'ows:Identifier')
6.     ows_Identifier.text = label
7.     # Append input items
8.     wps_DataInputs = SubElement(root, 'wps:DataInputs')
9.     for input in operation['inputs']:
10.         if len(input['value']) > 0:
11.             if input['type'] == 'geom':
12.                 wps_Input = SubElement(wps_DataInputs, 'wps:Input')
13.                 ows_Identifier = SubElement(wps_Input, 'ows:Identifier')
14.                 ows_Identifier.text = input['identifier']
15.                 if input['url'] == "":
16.                     wps_Data = SubElement(wps_Input, 'wps:Data')
17.                     wps_ComplexData = SubElement(wps_Data, 'wps:ComplexData')
18.                     wps_ComplexData.set('mimeType', 'application/json')
19.                     wps_ComplexData.text = input['value']
20.                 else:
21.                     wps_Reference = SubElement(wps_Input, 'wps:Reference')
22.                     wps_Reference.set('mimeType', 'application/json')
23.                     wps_Reference.set('xlink:href', input['value'])
24.                     wps_Reference.set('method', 'GET')
25.                 elif input['type'] == 'coverage':
26.                     wps_Input = SubElement(wps_DataInputs, 'wps:Input')
27.                     ows_Identifier = SubElement(wps_Input, 'ows:Identifier')
28.                     ows_Identifier.text = input['identifier']
29.                     wps_Data = SubElement(wps_Input, 'wps:Data')
30.                     wps_ComplexData = SubElement(wps_Data, 'wps:ComplexData')

```

```

31.                wps_ComplexData.set('mimeType', 'image/tiff')
32.                wps_ComplexData.text = input['value']
33.            else:
34.                wps_Input = SubElement(wps_DataInputs, 'wps:I
nput')
35.                ows_Identifier = SubElement(wps_Input, 'ows:I
dentifier')
36.                ows_Identifier.text = input['identifier']
37.                wps_Data = SubElement(wps_Input, 'wps:Data')
38.                wps_LiteralData = SubElement(wps_Data, 'wps:L
iteralData')
39.                wps_LiteralData.text = input['value']
40.            wps_ResponseForm = SubElement(root, 'wps:ResponseForm')
41.            wps_RawDataOutput = SubElement(wps_ResponseForm, 'wps:Raw
DataOutput')
42.            wps_RawDataOutput.set('mimeType', type)
43.            ows_Identifier = SubElement(wps_RawDataOutput, 'ows:Ident
ifier')
44.            ows_Identifier.text = 'result'
45.            url = operation['metadata']['url']
46.            headers = {'content-type': 'text/xml'}
47.            # Send the WPS execute's body to the WPS server for execu
tion
48.            r = requests.post(url, data=WorkflowUtils.prettify(root),
headers=headers)
49.            return r.text

```

The non-OGC compliant RESTful services don't require a similar implementation like the OGC WPS. This service category can be implemented by making an HTTP GET request to the REST endpoint with the specified parameters. To generate the complete URL for the RESTful service, we implemented a simple method shown in Listing 6.12 which loops through the inputs and builds the URL by appending the inputs to the REST endpoint.

Listing 6.12: Python Code for generating URL for RESTful web service.

```

1. def executeREST(operation):
2.     # Create a URL for the RESTful processing service
3.     endpoints = ""
4.     for input in operation['inputs']:
5.         endpoints = endpoints + "/" + quote(input["value"])
6.     url = operation["metadata"]["url"] + endpoints
7.     # Submit URL for execution
8.     results = requests.get(url)
9.     if results.text == "":
10.         results = []
11.     else:
12.         results = json.loads(results.text)
13.     return results

```

6.5.2. Process Chaining

Chaining processes is a useful feature of WPS which enables the creation of complex workflows from distributed web processing services. In Section 4.4, we discussed three approaches in which the Open Geospatial Consortium (2012) WPS standard 1.0 recommends chaining of services. These include:

1. Using BPEL engine to orchestrate services.
2. You are designing a WPS that calls other WPS processes in a sequence.
3. Cascading services chains as part of the execute request.

During our discussion, we identified weaknesses associated with each of the approaches based on our interests and guided by literature material of (Meek et al., 2016). Based on our discussion of the three approaches to process chaining, we identified that cascading service chains as part of the execute request in a workflow in the best option for this research. This approach implements a waterfall design concept used in software engineering where services follow a linear execution and the output of one service goes into the input of another service. To achieved linear execution, the correct sequencing of the services is required. In Section 2.2, we discussed an approach by Schäffer & Foerster (2008) to identify the sequencing of services by sorting the processes based on their topological relationships and using the three properties of DAG which include reflexivity, asymmetry, and transitivity. Following this discussion, we implement a recursive function for obtaining the topological sequencing of operations in a workflow using the connections of the operations specified in the workflow.

Listing 6.13: Recursive Function for Insertion Sort.

```

1. def recursiveF(connections, orderID, id):
2.     for connection in connections:
3.         if connection["toOperationID"] == id:
4.             if connection["fromOperationID"] in orderID:
5.                 orderID.remove(connection["fromOperationID"])
6.                 orderID.insert(0, connection["fromOperationID"])
7.                 WorkflowUtils.recursiveF(connections, orderID, connection["fromOperationID"])
8.     return orderID

```

The recursive function in Listing 6.13 above uses computer science insertion sort algorithm to identify if a source operation ID specified by *fromOperationID* is already in the sequence. If it is present, it removes it and makes it the first in the sequence since it has to be executed before the rest of the operations.

Listing 6.14: Code Snippet for Finding the Execution Order of Operations

```

1. def getExecutionOrder(workflow):
2.     operations = workflow["operations"]
3.     connections = workflow["connections"]
4.     # operIDs represent the IDs of all the operations
5.     operIDs = set()
6.     # NodeIDs represent the IDs of the parent operations
7.     nodeIDs = set()
8.     for operation in operations:
9.         operIDs.add(operation["id"])
10.        for connection in connections:

```

```

11.         if connection["fromOperationID"] == operation["id"]
12.             nodeIDs.add(operation["id"])
13.             break
14.         # leafIDs represents the IDs of the child operations
15.         leafIDs = list(operIDs.difference(nodeIDs))
16.         # orderID list the IDs of the operations in a sequential
17.         orderID = []
18.         orderID.extend(leafIDs)
19.         # Walk through the child operations to determine the parent
20.         for id in leafIDs:
21.             WorkflowUtils.recursiveF(connections, orderID, id)
22.         return orderID

```

The `getExecutionOrder()` function in Listing 6.14 is where the main sequencing takes place. Line 8-13 is a that walks through the workflow and picks the ID of the operations. The IDs are inserted to *operIDs* set. The operations which have at least a child node are stored in the *nodeIDs* set. To obtain the nodes without any child, we perform a set difference between the *operIDs* and *nodeIDs*. The result of this operation is stored in the *leafIDs* list. In the loop of lines 20-21, we implement a bottom-up approach to identify the parents of each childless node. The result of the ordered nodes is stored in the *orderID* list which is then used to chain the processes for execution.

6.5.3. Workflow Execution

Once we have determined the order of execution of the workflow processes, we implement a function which controls the execution by ensuring that the data flows sequentially from an output of one operation to the target operation. This method stores the output of each operation in a JSON object which is then submitted to the client once the whole workflow execution process is successfully terminated. In Listing 6.15 below, lines 11 to 13 are responsible for assigning the output of a previous operation to the target operation. When two operations are connected in the workflow client, the value assigned to the input of the target operation is specified by *fromOperationID_to_toParameterID*. For instance, the input value of “0_to_0” means that the parent operation has an ID zero (0) and its output is assigned to the first input of the current operation which has an ID of zero (0). Lines 16 and 24 determine whether the current operation requires a WPS or RESTful implementation. After that, a function responsible for that particular resource is called and executed. The implementation of these processing services was discussed in Sections 6.4 and 6.5.1.

Listing 6.15: Code Snippet for Executing the Workflow

```

1. def executeWorkflow(workflow):
2.     operations = workflow[0]["operations"]
3.     orderedIDs = WorkflowUtils.getExecutionOrder(workflow)
4.     outputs = {}
5.     j = 1
6.     result = []
7.     for id in orderedIDs:

```

```

8.         operation = WorkflowUtils.getOperationByID(id, operations)
9.         if len(outputs) > 0:
10.             for i in range(0, len(operation["inputs"])):
11.                 if "_to_" in operation["inputs"][i]["value"]:
12.                     value = operation["inputs"][i]["value"].split("_to_")
13.                     operation["inputs"][i]["value"] = outputs[
                        value[0]][0]
14.
15.             output = ""
16.             if operation["metadata"]["resource"] == "WPS":
17.                 if operation['outputs'][0]['type'] == "geom":
18.                     output = WorkflowUtils.executeWPS(operation,
19. 'application/json')
20.                 elif operation['outputs'][0]['type'] == "coverage":
21.                     output = WorkflowUtils.executeWPS(operation,
22. 'image/tiff')
23.                 else:
24.                     output = WorkflowUtils.executeWPS(operation)
25.
26.             if operation["metadata"]["resource"] == "REST":
27.                 output = WorkflowUtils.executeREST(operation)

```

The result of the workflow execution is sent to the client as a JSON object. Each operation is assigned the path to its output data which the user can use to download the data. Our implementation allows users to view the result of their execution using the map panel. For raster data formats, the user is required to specify the settings for the GeoServer where the data is to be published to allow rendering of the map using WMS. This can be achieved using the *PublishRaster* RESTful service which we implemented. Vector data does not require the use of GeoServer since we can render the map as a layer using the GeoJSON data format.

6.6. Workflow Transformation

To further reinforce our concept of enhancing shareability and reproducibility of geoprocessing workflows, we implement a method for transforming workflows from one WfMS to another. This method is based on our discussions in chapter three and chapter five where we proposed a platform-independent workflow interchange schema and a framework for mapping constructs between different WfMSs. An implementation of the algorithm for the discovery of corresponding processes in different GIS tools which were discussed in Section 5.2 is also carried out in this section. Figure 6.4 illustrates the workflow transformation that has been implemented for this demonstration using a BPMN, ILWIS and QGIS workflows. The platform-independent workflow interchange schema act as a link between the workflow schema of the different WfMSs which is a similar concept as the Model Driven Architecture (MDA) transformations. For instance, to share an ILWIS workflow with a BPMN compliant WfMS, the

ILWIS workflow format goes through two transformations. First, it is transformed into the platform-independent interchange format using the proposed standard schema. The second transformation converts the platform-independent interchange format to a BPMN document.

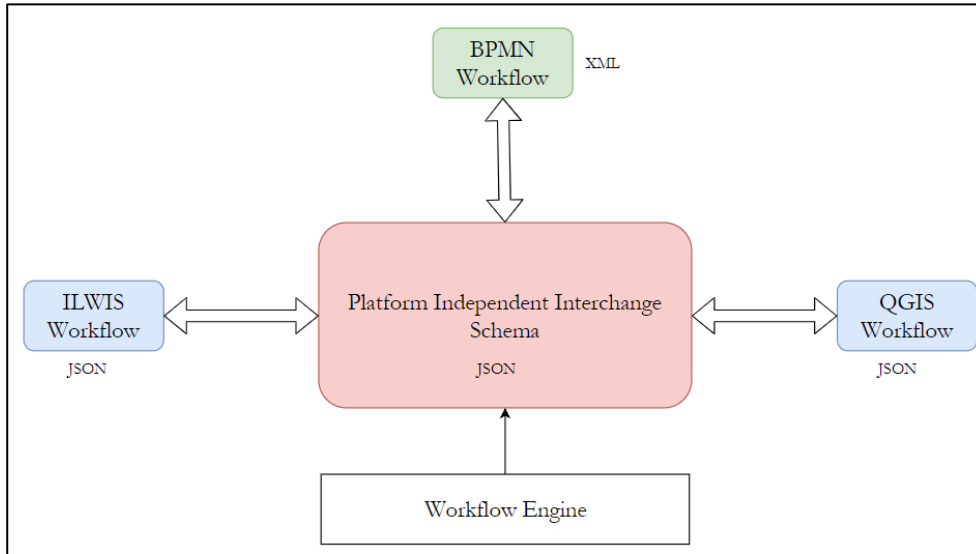


Figure 6.4: Workflow Transformation

6.6.1. Sharing Workflows in BPMN Compliant WfMSs

We implement two-pair functions for transforming workflows from one representation format to another which uses the schema of different interchange formats to map related constructs and keywords of different WfMSs shown in Table 5.1. The code snippets in the Listings 6.16 and 6.17 below illustrate the function used to transform a platform-independent interchange to a BPMN document. The function first creates the root element which is composed of the BPMN schema. From the root element, we implement sub-elements where the top-most sub-element is the process which corresponds to the workflow. We set the id and process name in 17 and 19 of Listing 6.16 which correspond to the workflow id and the longname. Line 23 to 31 are used to define the sequence flows which correspond to connections between different operations in the workflow.

Listing 6.16: Code snippet for initializing the process element and sequence flows (connections).

```

1. def pimToBPMN(workflow):
2.     """
3.     This function trasform a workflow representation from the JSON-
4.     based platform indipendent model to
5.     XML-based BPMN document
6.     :return: Generated XML-
7.     based BPMN document which can be opened with any BPMN tool
8.     """
9.     # Initialize the root element of the BPMN document
10.    root = WorkflowUtils.bpmnHead(WorkflowUtils)
11.    itemDefinition = SubElement(root, 'bpmn2:itemDefinition')
12.    itemDefinition.set("id", "ITEM_DEF_STRING")
13.    itemDefinition.set("isCollection", "false")
14.    itemDefinition.set("structureRef", "xs:string")
15.
16.    # Set the process element
  
```

```

15.         process = SubElement(root, 'bpmn2:process')
16.         # process id corresponds to the JSON-based workflow id
17.         process.set("id", "_" + str(workflow["id"]))
18.         # process name corresponds to the JSON-
        based workflow longname
19.         process.set("name", workflow["metadata"]["longname"])
20.         process.set("isExecutable", "true")
21.
22.         # SequenceFlow represent the connections between tasks
23.         sequenceFlow = SubElement(process, 'bpmn2:sequenceFlow')
24.
25.         sequenceFlow.set("id", "SequenceFlow_Start")
26.         sequenceFlow.set("sourceRef", "StartEvent_1")
27.         # The target node of the start event is the first task in
        the execution order
28.         sequenceFlow.set("targetRef", "ServiceTask_" + str(Workfl
        owUtils.getExecutionOrder(workflow)[0]))
29.         i = 1
30.         for connection in workflow["connections"]:
31.             sequenceFlow = SubElement(process, 'bpmn2:sequenceFlo
        w')
32.             sequenceFlow.set("id", "SequenceFlow_" + str(i))

```

The code snippet in *Listing 6.17* was used to create serviceTasks and dataInputs for BPMN document. Line 5 creates the identifier and line 7 sets the name tag for the serviceTask. These correspond to the operation's id and longname in the JSON format respectively. The implementation url was set in line 9 which also corresponds to the URL in the metadata object of the JSON format. Lines 15-22 are responsible for appending dataInputs to the BPMN document.

Listing 6.17: Code snippet for creating service tasks and data inputs.

```

1. for id in WorkflowUtils.getExecutionOrder(workflow):
2.     operation = WorkflowUtils.getOperationByID(id, operations)
3.     task = SubElement(process, 'bpmn2:serviceTask')
4.     # Service Task id corresponds to the operations id
5.     task.set("id", "ServiceTask_" + str(operation["id"]))
6.     # Service Task name corresponds to the operation's longname
7.     task.set("name", operation["metadata"]["longname"])
8.     # The implementation engine of the service corresponds to the W
        PS/REST endpoint
9.     task.set("implementation", operation["metadata"]["url"])
10.    task.set("resource", operation["metadata"]["resource"])
11.    ioSpecification = SubElement(task, 'bpmn2:ioSpecification
        ')
12.    ioSpecification.set("ioSpecification_", "ioSpecification_
        " + str(id))
13.    inputSet = SubElement(ioSpecification, 'bpmn2:inputSet')
14.    # BPMN dataInput is mapped to the inputs of an operation
15.    for input in operation["inputs"]:
16.        dataInput = SubElement(ioSpecification, 'bpmn2:dataIn
        put')
17.        dataInput.set("id", "DataInput_" + input["name"] + "_"
        " + str(id))
18.        dataInput.set("itemSubjectRef", "ITEM_DEF_STRING")
19.        dataInput.set("name", input["name"])

```



```

20.         dataInput.set("type", input["type"])
21.         dataInput.set("optional", str(input["optional"]).lower())
22.         dataInput.set("value", input["value"])

```

To establish connections between processes in the process chain, we used the code snippet in Listing 6.18. The keyword *fromOperationID* corresponds to BPMN *sourceRef* while *toOperationID* corresponds to *targetRef*. Transformation of the JSON-based platform independent interchange format of the workflow produces a BPMN document whose extract shows a sequence flow similar to the one in Listing 6.19.

Listing 6.18: Code snippet for mapping JSON connections to BPMN serviceFlows.

```

1. for connection in workflow["connections"]:
2.     sequenceFlow = SubElement(process, 'bpmn2:sequenceFlow')
3.     sequenceFlow.set("id", "SequenceFlow_" + str(i))
4.     if i == 1:
5.         sequenceFlow.set("sourceRef", "ServiceTask_0")
6.     else:
7.         sequenceFlow.set("sourceRef", "ServiceTask_" + str(connection["fromOperationID"]))
8.
9.     sequenceFlow.set("targetRef", "ServiceTask_" + str(connection["toOperationID"]))
10.    if i == len(workflow["connections"]):
11.        sequenceFlow = SubElement(process, 'bpmn2:sequenceFlow')
12.        sequenceFlow.set("id", "SequenceFlow_End")
13.        sequenceFlow.set("sourceRef", "ServiceTask_" + str(i))
14.        sequenceFlow.set("targetRef", "EndEvent_1")

```

Listing 6.19: An extract of a BPMN sequenceFlow for a simple workflow.

```

1. <bpmn2:process id="_1" name="Subworkflow" isExecutable="true">
2.     <bpmn2:sequenceFlow id="SequenceFlow_Start" sourceRef="StartEvent_1" targetRef="ServiceTask_2" />
3.     <bpmn2:sequenceFlow id="SequenceFlow_1" sourceRef="ServiceTask_0" targetRef="ServiceTask_1" />
4.     <bpmn2:sequenceFlow id="SequenceFlow_2" sourceRef="ServiceTask_2" targetRef="ServiceTask_3" />
5.     <bpmn2:sequenceFlow id="SequenceFlow_3" sourceRef="ServiceTask_4" targetRef="ServiceTask_5" />
6.     <bpmn2:sequenceFlow id="SequenceFlow_4" sourceRef="ServiceTask_1" targetRef="ServiceTask_5" />
7.     <bpmn2:sequenceFlow id="SequenceFlow_5" sourceRef="ServiceTask_3" targetRef="ServiceTask_5" />
8.     <bpmn2:sequenceFlow id="SequenceFlow_End" sourceRef="ServiceTask_5" targetRef="EndEvent_1" />
9.     <bpmn2:startEvent id="StartEvent_1" name="Start Workflow">
10.         <bpmn2:outgoing>SequenceFlow_Start</bpmn2:outgoing>
11.         <bpmn2:outgoing>SequenceFlow_1</bpmn2:outgoing>
12.     </bpmn2:startEvent>
13.     <bpmn2:endEvent id="EndEvent_1" name="End Workflow">
14.         <bpmn2:incoming>SequenceFlow_End</bpmn2:incoming>

```

```
15.      </bpmn2:endEvent>
```

6.6.2. Sharing Workflows in Non-Standardization Compliant WfMSs

Transformation of workflows from the platform-independent JSON schema to a BPMN document misses one crucial step which is important to illustrate reproducibility of workflows in other WfMSs which do not implement web services. This is because these WfMSs discussed in Section 3.3 completely rely on their process engines and internal environment to successfully execute a workflow. As a result of this, there is a need to adopt their corresponding process names when sharing workflows. We implement two approaches to support sharing of workflows in non-standardization compliant WfMS which make it possible to use platform-specific process names and their definitions in terms of input and output requirements.

The first approach occurs at the client-side of the developed application and involves using the property window of the operation in the workflow editor panel. The user selects the endpoint for the processing server in the drop-down select box. The selected endpoint loads new operations in the drop-down list as shown in Figure 6.5. When a user selects the corresponding operation, the visual object for the process is redrawn with the new definition for the selected operation.

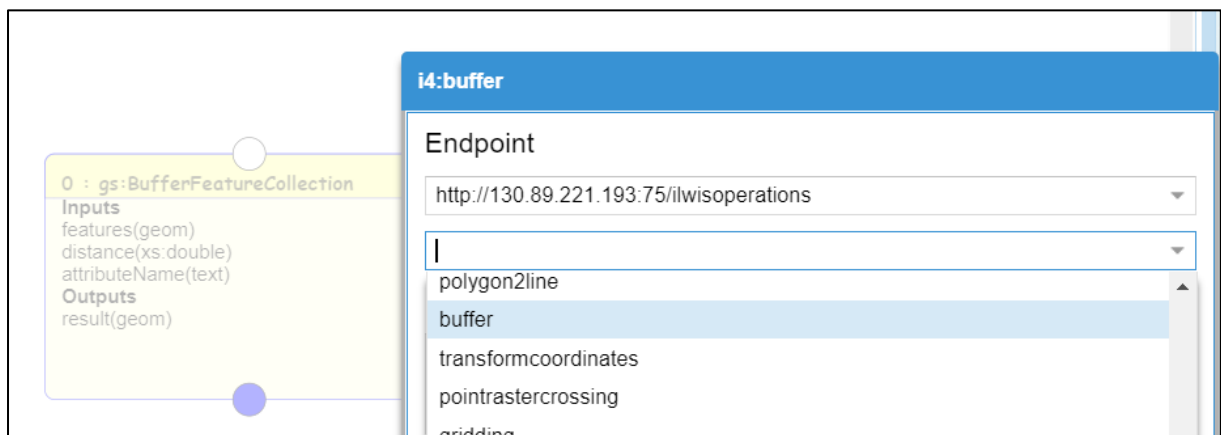


Figure 6.5: Changing resource providers for the same process

The second approach takes place at the server-side of the application and is achieved using the workflow engine. This approach initiated when a user chooses to export their workflow to QGIS or ILWIS workflow interchange format. A python script was developed that takes a JSON representation of the workflow and transform it into the interchange format of the target WfMS. The first step in the transformation involved the implementation of the flowchart in Figure 5.3 to help in the discovery of corresponding process names in the target software. The code snippet in Listing 6.20 was used to search for operations based on a search string obtained from the provenance information in the workflow. Line 7 is responsible for reading the JSON file where the operations for each GIS tool is stored. Line 8 filter only the operations of the target GIS tool which the user is interested in. In lines 13 to 15, we implement a

loop which walks through the keywords in the operations list and checks if a keyword matches the search string. In case of a positive match, the *count* of hits is incremented. Lines 16 to 18 assign the maximum hits to the *max* variable and the matching operation to *oper* variable.

Listing 6.20: Code snippet for searching an operation based on a keyword

```

1. def searchOperation(tool, searchString):
2.     """
3.     :param tool: GIS tool owning the operation
4.     :param searchString: Keyword for the search
5.     :return: Return the operation with the highest hit
6.     """
7.     json_data = open("operations.json").read()
8.     operations = json.loads(json_data)[tool]
9.     max = 0
10.    oper = None
11.    for operation in operations:
12.        count = 0
13.        for keyword in operation["keywords"]:
14.            if keyword in searchString:
15.                count = count + 1
16.            if count > max:
17.                max = count
18.                oper = operation
19.    return {"hits": max, "operation": oper}

```

After finding the best matching corresponding operation of the target WfMS, we use its internal name, input and output parameter requirements in mapping the workflow from the platform-independent interchange format to the specific interchange format. For instance, the code snippets in Appendix E helps in transforming platform-independent workflow (PIW) to a QGIS workflow (PSW) format which can be visualized and executed using QGIS WfMS. To illustrate the transformation, a simple workflow involving two GeoServer operations *gs:Centroid* and *gs:Buffer* was created using the generic workflow client. The corresponding internal process names in QGIS are *qgis:polygoncentroids* and *gdalogr:bufferectors* respectively. The successful transformation between the two WfMS led to mapping from the visual workflow in A to B as shown in Figure 6.6. The resulting workflow in QGIS WfMS can be executed to produce the same result thereby ensuring reproducibility.

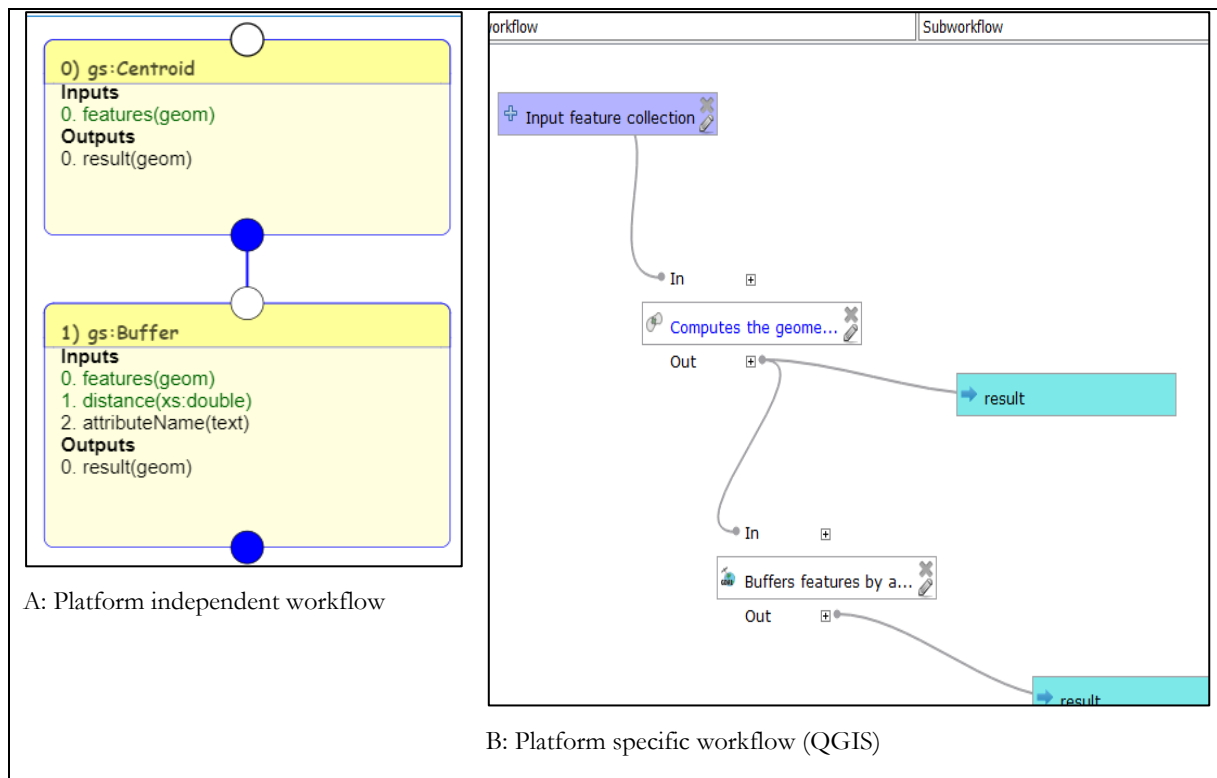


Figure 6.6: Transformation of PIW to QGIS Workflow

7. PROOF OF CONCEPT

Having discussed in the previous chapters the theoretical concepts on workflows and the implementation of a prototype system that facilitates sharing and reproduction of workflows, in this chapter, we demonstrate a proof of concept based on a use case in the AfriAlliance project. The AfriAlliance project aims to “prepare Africa for future climate change challenges by creating the opportunity for African and European stakeholders to work together in the areas of water innovation, research, policy, and capacity development” (Mannaerts et al., 2017a). One of the approaches towards achieving their aim is the use of a triple-sensor approach to improve information gathering for water resource monitoring and forecasting. Water resource monitoring entails the provision of adequate qualitative and quantitative information about the state of the water resource at any moment (Garcia et al., 2016). Getting the latest and accurate information for water resource monitoring or disaster management is a challenge with many satellite products and in-situ generated data. This is because of the low temporal and spatial resolution of these data sources. The triple-sensor approach combines three mutually independent data sources which include space-based satellite sensors, human sensors (crowdsourced geoinformation) and physical in-situ sensors (meteorological stations). Human sensor information is the latest source of geospatial data driven by advancements in technology. Consumption of data generated by humans is becoming more popular because it is more recent and provide precise information. With the latest progress in technology, a large amount of heterogeneous and distributed geospatial data is becoming available. As a result, scientists are faced with the challenges of combining these data to solve specific problems. One of the challenges lies in its accessibility, reliability, and accuracy. Scientists have developed varying opinions for their choice of the data sources where some prefer satellite to in-situ data.

On the other hand, most community-based projects would prefer to use crowdsourced geoinformation. Through the triple sensor sensors approach, AfriAlliance propose a triple collocation method which is derived from the observation that a particular data source would provide more reliable and accurate information at a particular location as compared to others. In the following sections, we discuss factors that affect the combination of the three sources of geospatial data and the triple collocation approach. We then compose a workflow from a determined set of web services to be used in triple collocation.

7.1. Satellite, In-situ and Crowdsourced Geoinformation

Remote sensing technology and in-situ measurements observed from local weather stations are the two traditional sources of geospatial data that have extensively contributed to scientific research. One of the scientific application of data obtained from these sources has been in the management of water resources. For instance, in monitoring the growth of the harmful algae blooms in recreational water bodies and drinking water (Clark et al., 2017), evaluation of extreme precipitations for water resource and flood risk management (Dhib et al., 2017). Better water resource management is critical to helping people,

economies, and ecosystems to thrive, reduce poverty and sustain prosperity. However, successful water management requires detailed knowledge of the available water resources which can only be achieved through effective monitoring and forecasting. The last decade has seen the emergence of a third data stream (crowdsourced geoinformation) where humans are involved in scientific research by creating and sharing information. Combining the three sources of data helps eliminate their limitations thereby providing accurate and reliable information for effective water resource management. We discuss the following properties that are relevant for the effective combination of data from the three sources in the triple sensor approach. These findings are based on the report by (Mannaerts et al., 2017a).

i. Data Variable

This represents the observed climate or water variable which can include surface water level, soil moisture, precipitation amount, vegetation condition, temperature, etc. The chosen data variable should be the same for all three sensors.

ii. Data representation format

Another important consideration for combining satellite, in-situ and crowdsourced geoinformation is the representation format for the data. The same representation format should be used for all three data sources. For instance, combining a Boolean and nominal variable does not yield positive results. The data type used in representing the data is crucial for the successful application of the triple sensor approach.

iii. Temporal collocation

The period for which the sampling has been carried out should be the same for all the three sources of data. For the satellite data, the sampling period can be affected by the temporal resolution of the satellite. Most in-situ stations reporting is done regularly which can occur at an hourly or daily basis. This is different from citizen observations which may not be done at regular intervals. Matching of the periods for three data sources is necessary for effective comparison and validation.

iv. Spatial collocation

The observed data from the three sources must be occupying the same geographical space to be able to align them. Low spatial resolutions for satellite products and low density of in-situ stations affect the combination of these data sources. In as much as the crowdsourced geoinformation is increasingly becoming available due to the growth in technology, the density of meteorological stations is still inadequate for most parts of Africa. There has been an effort by private organizations to fill in-situ data with their measurements. However, this again does not cover the whole African continent.

v. Coordinate System

For effective alignment, the data must be in the same coordinate system. In case the data are of the different coordinate system, then they must be projected or transformed into one coordinate system. However, if this is not done the triple sensor approach fails.

vi. Data Quality

The success of the triple-sensor approach in the validation of data for water resource monitoring lies significantly in the quality of the observed data. Low quality of data is mostly attributed to citizen-based observations. However, intensive capacity development, volunteer engagement activities, and incentives, as well as effective infrastructure for data collection can provide reliable citizen-based data water monitoring and forecasting.

7.2. Triple Collocation

Triple collocation (TC) is a method based on statistical covariance that is used to estimate the unknown error standard deviations or RMSE of three independent data sources to determine their reliability (McColl et al., 2014). Since the three data sets are mutually independent, TC assumes that each of them has its errors which are introduced from their measurements and there is no systematic bias among them. The following equation gives the error model for TC.

$$x_i = \alpha_i + \beta_i T + \varepsilon_i$$

The x_i is the collated observation from the i^{th} measurement system ($i \in \{1,2,3\}$). The observation x_i is linearly related to the true value T with an additive random error ε_i . The measurement systems in the case of triple sensor represents satellite, in-situ and human sensors. TC further determines the covariance for the three measurements by applying a formula proposed by McColl et al. (2014) which has been used to estimate correlation coefficients for three independent data sets in various scientific studies. Since TC assumes that the systematic errors from the three measurements are not related, their covariance is zero. At this point, we cease discussing more about the entire formula of triple collocation since it has already been implemented as a process in one of the GIS tools and we can reuse it in our workflow. However, in addition to the references made in this Section, we point out to other literature materials on the same such as (McColl et al., 2016) and (Li et al., 2017).

Triple collocation has been widely used in scientific research to estimate errors in measurements. For instance, it was used by Leroux et al. (2011) to compare the performance of soil moisture satellite products; Soil Moisture and Ocean Salinity (SMOS), Advanced Microwave Scanning Radiometer-Earth Observing System (AMSR-E) and Advanced Scatterometer (ASCAT). TC has also been used to assess the accuracy of classifications in case of earthquake damage assessment without the relying on ground truth (Pierdicca et al., 2018).

We found TC to be relevant to this research because it provides a framework in which we can combine satellite data, in-situ and crowdsourced geoinformation in a workflow using web services which we discussed in Chapter 4. Satellite data are accessed using WCS whereas in-situ and crowdsourced geoinformation are accessed using SOS. ILWIS has already implemented a geoprocessing function for triple collocation making it easier to integrate the process as web service in a workflow.

7.3. Shareable and Reproducible Workflow for Triple Collocation

7.3.1. Study Area

The study area for this demonstration is in the southwestern region of Burkina Faso, a town called Dano which is a research area of West African AfriAlliance partner, West African Science Service Centre on Climate Change and Adapted Land Use (WASCAL)¹⁵.

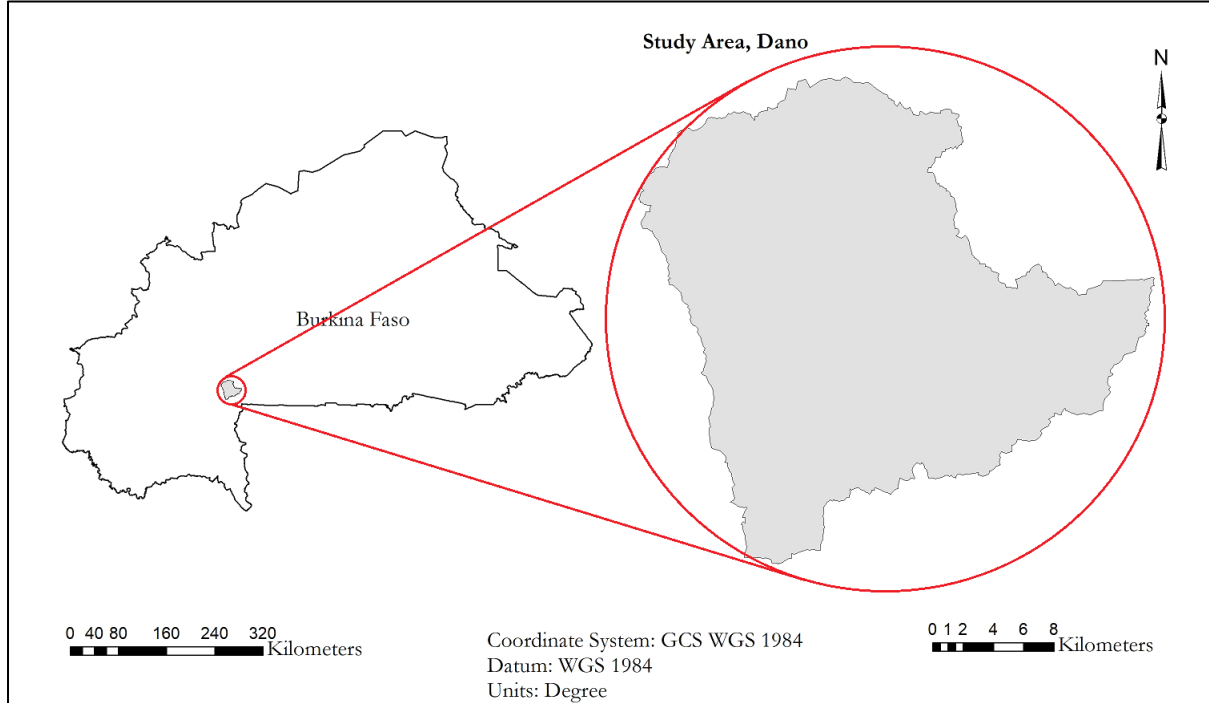


Figure 7.1: Study Area, Dano Burkina Faso.

7.3.2. Data

For this illustration, we used three sources of spatial data mainly satellite, ground stations, and citizens for July 2015. The abstract workflow in Figure 7.3 indicates three sources for satellite-based rainfall data that can be used for triple collocation, however for verification of our results with the work of Mannaerts et al. (2018) we used CHIRPS¹⁶ rainfall product. The source of in-situ station data is NOAA Climate Prediction Centre¹⁷ (CPC) while the crowdsourced geoinformation was obtained from the Water Point Data Exchange database¹⁸ (WPDE) which uses citizen-based data collection methods to collect information on water points status across the globe.

We implemented a web coverage service (WCS) for the CHIRPS rainfall products which provides accessible and shareable links for the raw data in a GeoTIFF format. A sensor observation service (SOS) was implemented for the ground station and citizen data obtained from NOAA CPC and WPDE respectively. This was done by creating a database of all the data observations from in-situ and citizens

¹⁵ <http://www.wascal.org/about-wascal/welcome-to-wascal/>

¹⁶ <ftp://chg-ftpout.geog.ucsb.edu/pub/org/chg/products/CHIRP/>

¹⁷ ftp://ftp.cpc.ncep.noaa.gov/precip/CPC_UNI_PRCP/GAUGE_GLB/V1.0/

¹⁸ <https://www.waterpointdata.org/water-point-data>

and then using a python script to implement the OGC specification for SOS. By default, the SOS GetObservation requests were implemented to return a GeoJSON output data format. Retrieval of XML data format depending is also possible depending on the output format specified by the users. Specifying the period for the event allows execution of the workflow with data for the observed time interval.

GetObservation	<pre>http://WWW.EXAMPLE.COM/WorkflowApp/app/api/sos.py? service=SOS&request=GetObservation& version=1.0.0&observedProperty=Rainfall_sensors& offering=rainfall_SENSORS& eventTime=2015-07-01/2019-01-30T22:36:42&outputFormat=json</pre>
----------------	--

The implementation allows time-series inspection of in-situ and human sensor data by clicking on the point of interest in the map. The time-series data for a selected point is plotted in a line-chart as shown in Figure 7.2.

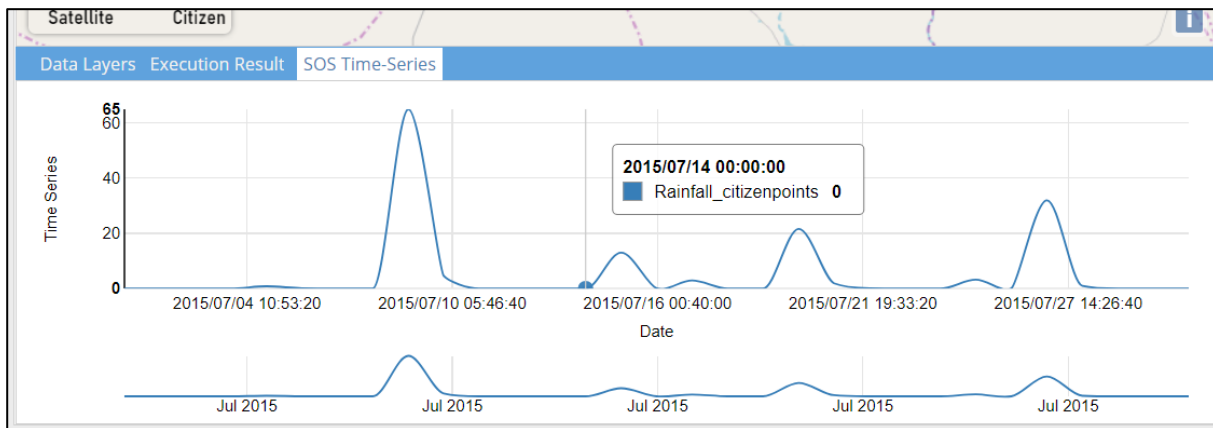


Figure 7.2: Time-series Analysis of Sensor Data.

7.3.3. Method

This demonstration adopted the abstract workflow, shown in Figure 7.3 below, for triple collocation which was produced by (Mannaerts et al., 2017b). The abstract workflow provides an overview of the operations, their input and output and hides the implementation details. Accumulated precipitation for satellite data is derived from either CHIRPS, TAMSAT or RFE. In the previous chapters, we discussed that to create a sharable and reproducible workflow, it must be composed of web services (Chapter 4) and also follow a standard schema (Chapter 5) which make it possible to transform to other workflow representation formats.

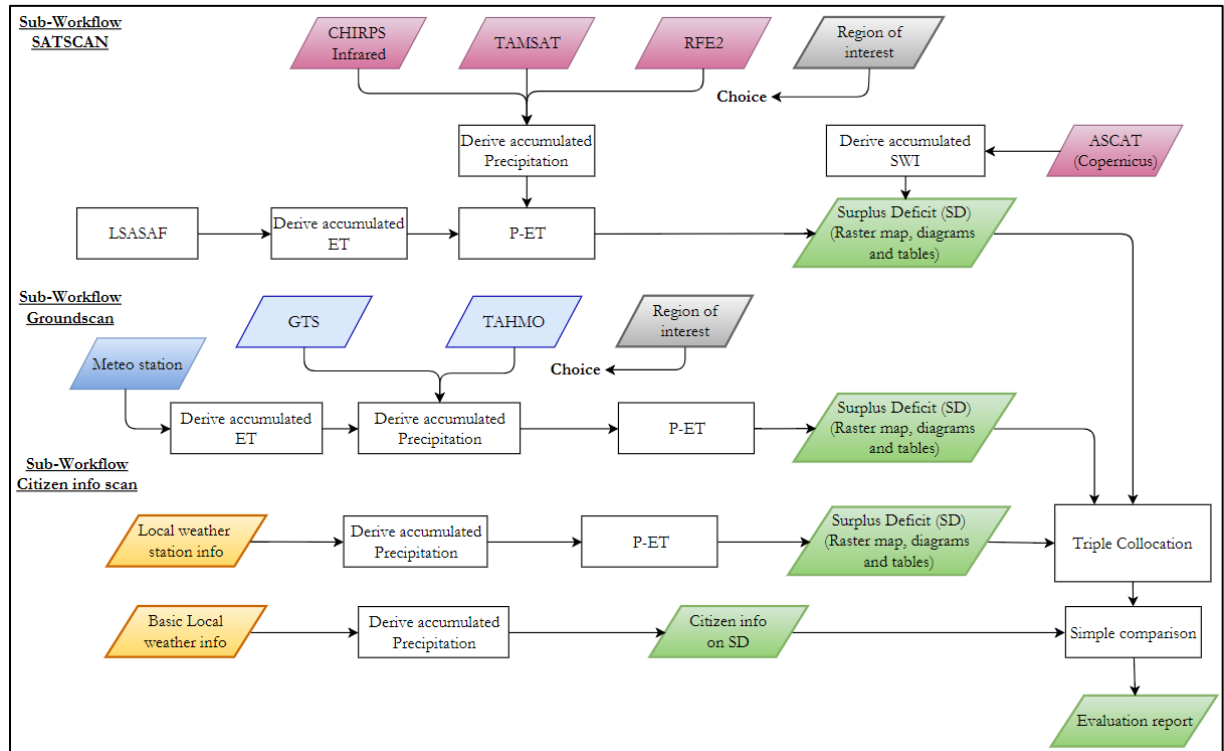


Figure 7.3: Abstract Workflow for the Triple Sensor Water Accounting. Source: (Mannaerts et al., 2017a)

Our analysis of the abstract workflow leads to the creation of a corresponding concrete workflow as shown in Figure 7.4. From the concrete workflow, six (6) operations were identified which are required for the triple sensor water accounting workflow. Our implementation makes use of ILWIS operations only since we could not find corresponding operations exposed as web services for other GIS tools. The moving average operation is required to interpolate in-situ data for a specified georeference. Our implementation of the moving average obeys the operation input requirements for ILWIS. However, we use a python script to iterate over the attributes and execute the moving average operation for each specified attribute. The attributes of moving average operation are obtained from the attribute table of the in-situ data which represents the observed dates, and we separate them by a semi-colon. We then create a map list from the resulting maps of moving average. We use the table operations to create a point map from the citizen-generated data. This is a two-step process which requires first creating an ILWIS table from the GeoJSON data format obtained from the SOS GetObservation request. The second step creates a point map from the resulting table. We implemented a second web processing service which uses ILWIS raster operation to create a map list of available rainfall data for a selected period. The map lists of satellite and in-situ data, together with the point map of citizen-generated data are then passed as inputs to the triple collocation operation. Upon successful execution, the result of this workflow is an evaluation report for triple collocation which is a point map. Though abstracted in the workflow, we use a python script to create a GeoJSON from the point map result of triple collocation which we use for visualization in the geoportal.

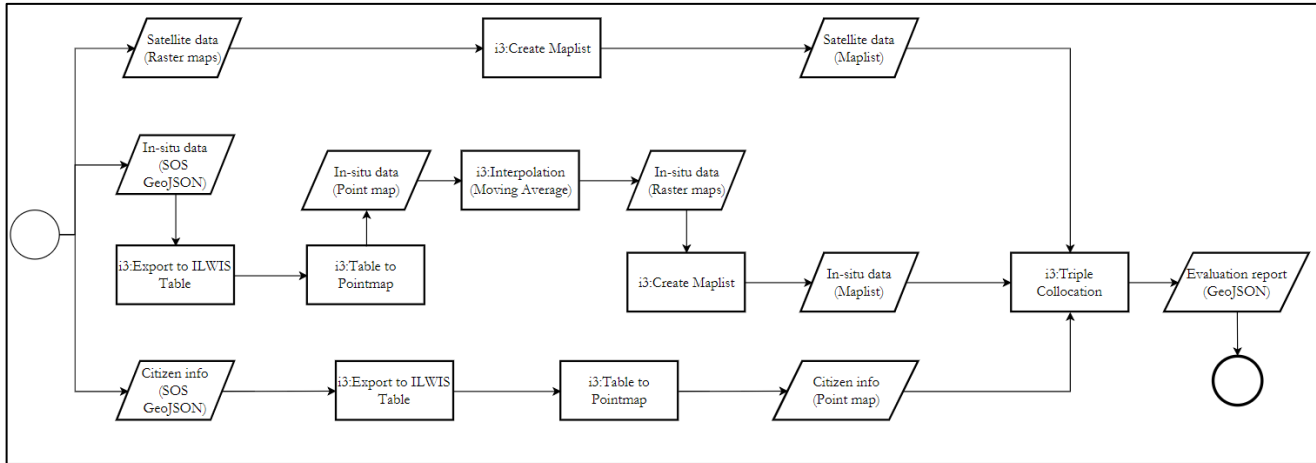


Figure 7.4: Concrete Workflow for Triple Sensor Approach.

Since ILWIS also does not expose their operations as web services, we implemented web processing services from these ILWIS operations using the OGC WPS specifications. These web services run on an ILWIS engine which is driven by ILWIS objects. Using these web processing services together with WCS and SOS, we compose a workflow using the standard schema we discussed in Chapter 5. The visual representation of the workflow is as shown in Figure 7.5 while an extract from its corresponding textual representation is as shown in Figure 7.6.

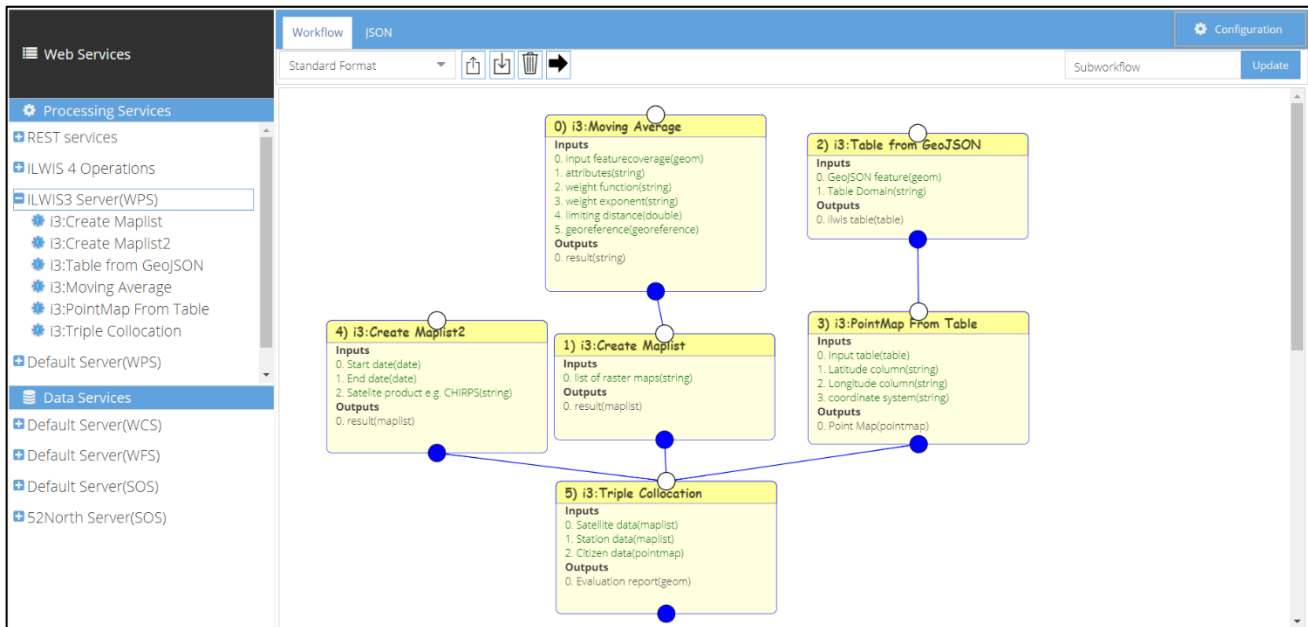
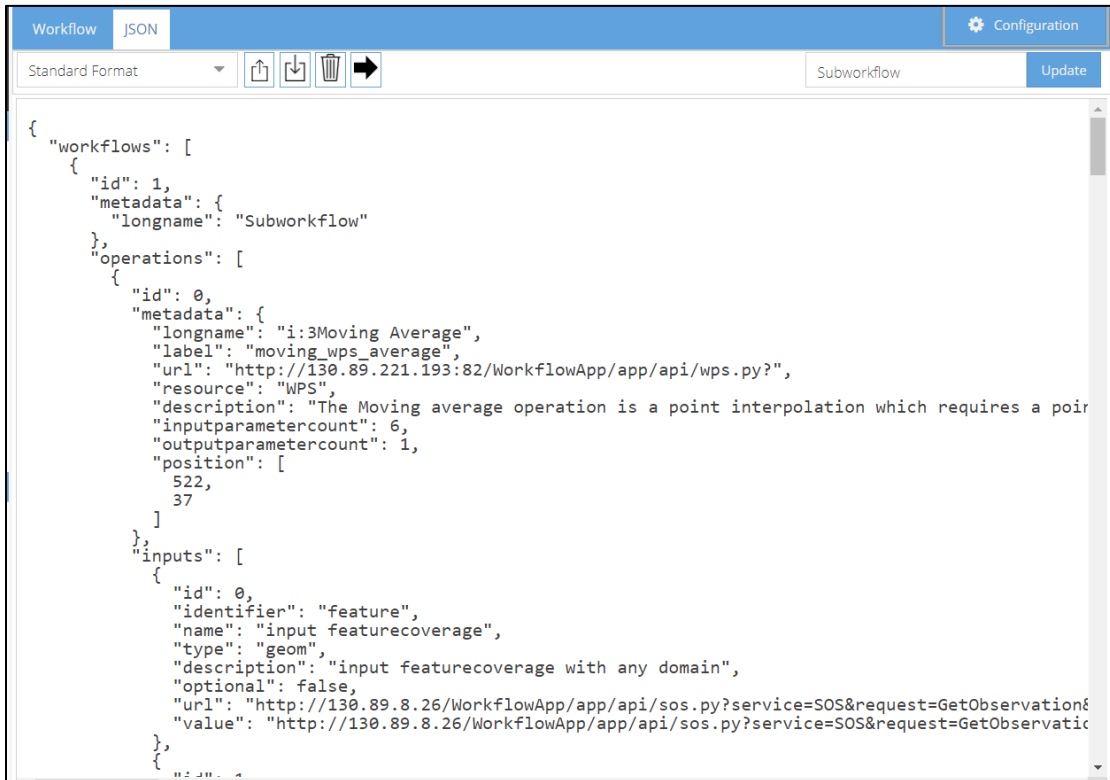


Figure 7.5: Triple Sensor Workflow Composition from Web Services.



```

{
  "workflows": [
    {
      "id": 1,
      "metadata": {
        "longname": "Subworkflow"
      },
      "operations": [
        {
          "id": 0,
          "metadata": {
            "longname": "i:3Moving Average",
            "label": "moving_wps_average",
            "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
            "resource": "WPS",
            "description": "The Moving average operation is a point interpolation which requires a point",
            "inputparametercount": 6,
            "outputparametercount": 1,
            "position": [
              522,
              37
            ]
          },
          "inputs": [
            {
              "id": 0,
              "identifier": "feature",
              "name": "input featurecoverage",
              "type": "geom",
              "description": "input featurecoverage with any domain",
              "optional": false,
              "url": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=GetObservation&",
              "value": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=GetObservation&"
            }
          ]
        }
      ]
    }
  ]
}

```

Figure 7.6: JSON extract of the Triple Sensor Workflow.

The textual representation of the workflow is shareable and reproducible for other WfMSs such as ILWIS. Our implementation allows transformation of the textual representation to an XML based BPMN document which can be shared and reproduced in BPMN compliant tools. We demonstrated the concept by sharing the workflow with Camunda modeler as shown in Figure 7.7. The complete JSON and XML textual representations for the Triple Sensor workflow can be observed in the Appendix Section.

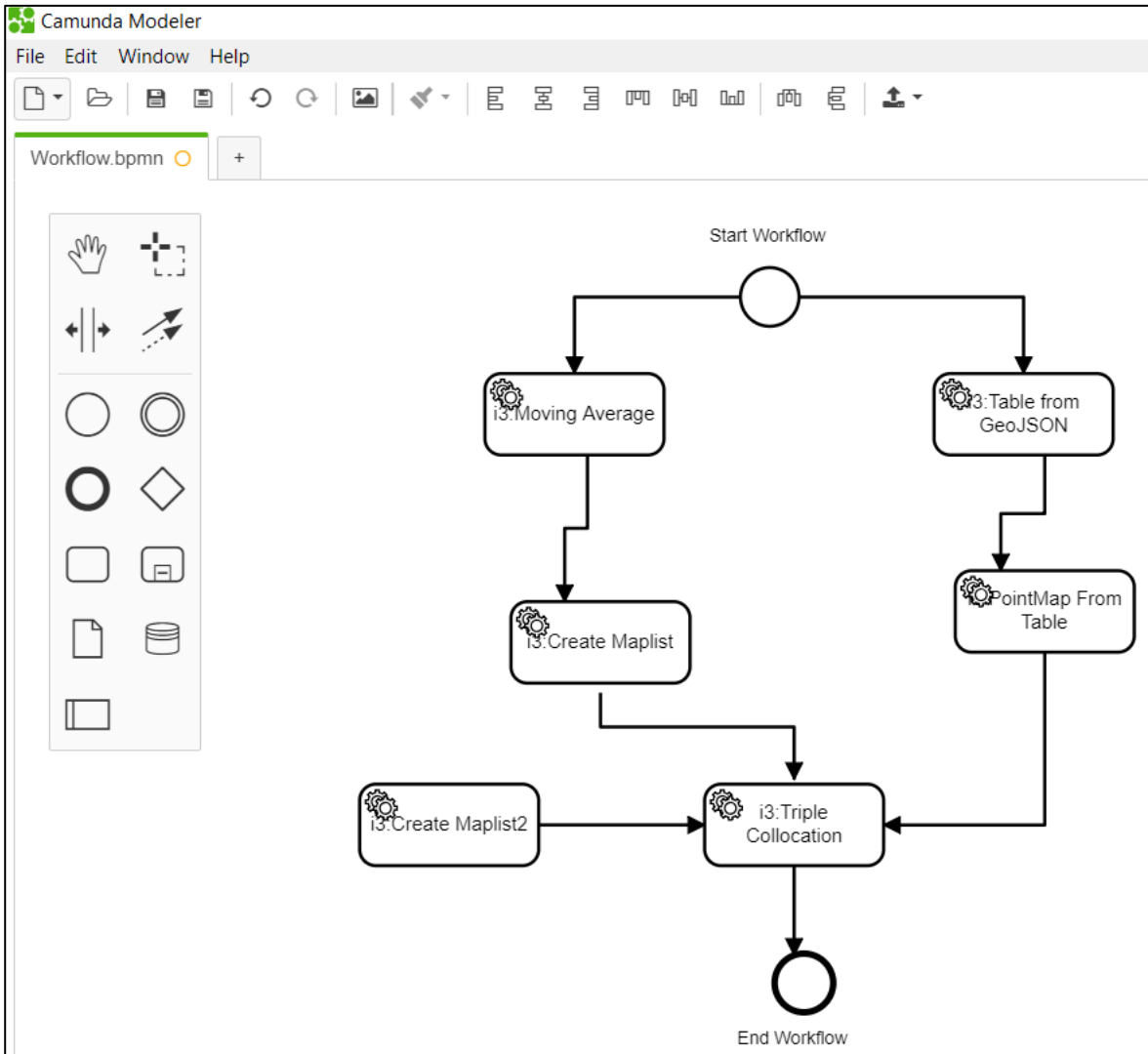


Figure 7.7: Visualization of BPMN-based Triple Sensor Workflow in Camunda modeler.

7.4. Result Discussion

The execution of triple collocation workflow realizes an evaluation report indicating the strengths and performance of each of the three sources of data at specific locations. The report is visually illustrated in Figure 7.8 below. Hovering the mouse over the points on the map, open a popup window which displays the various attributes. The attributes w_1 , w_2 , and w_3 correspond to the weights assigned to each of the three sources data used as inputs in the triple collocation. For instance, w_1 corresponds to the weight assigned to the satellite sensor, w_2 corresponds to in-situ sensors whereas w_3 corresponds to citizen sensors. A data source with the highest weight indicates a high confidence level associated with it for that particular location. The report also indicates the error variances for each weight. The color symbology assigns the color blue to high weights associated with satellite data (w_1), red is associated with high weights for in-situ data (w_2) whereas green is associated with high weights for citizen data (w_3).

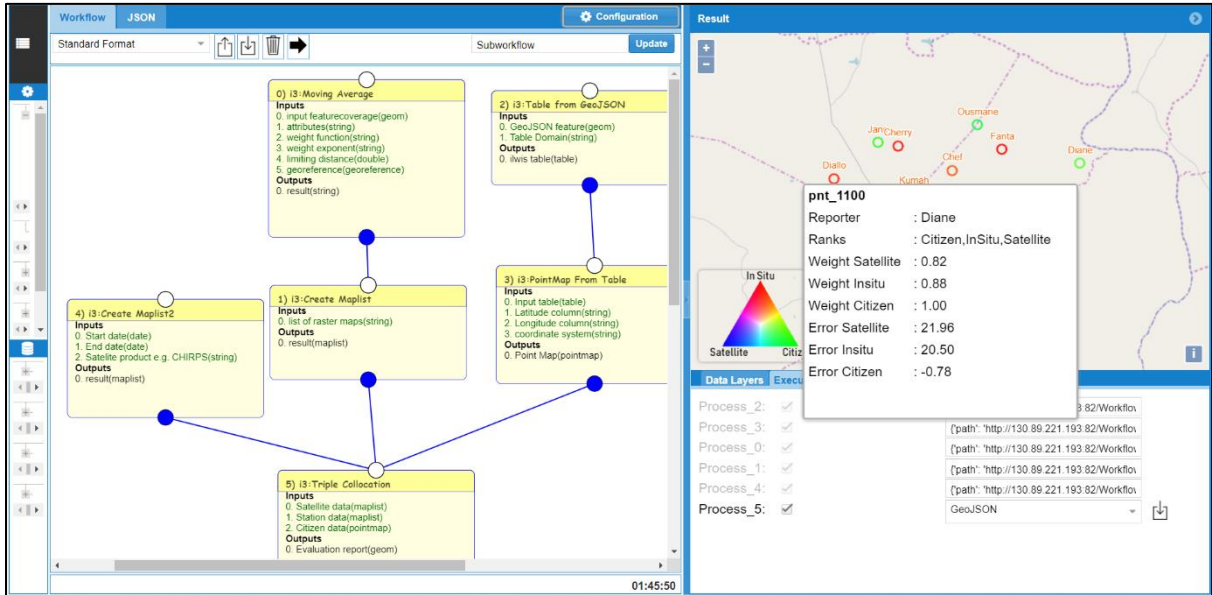


Figure 7.8: Result Analysis of Triple Sensor Workflow Execution.

The result of the execution of Triple Sensor workflow using our approach is compared to the findings by Mannaerts et al. (2018) in Table 7.1. The result from our approach is listed in the columns by abbreviation **A** while the ones from Mannaerts et al. (2018) by **B**. From the observation of both findings, in-situ sensor records the best performance in five (5) stations out of the eleven (11) used in this demonstration, the citizen sensor five (5) whereas satellite sensor one (1).

Table 7.1: Comparison of results from the Triple Sensor Workflow to findings by Mannaerts et al. (2018)

Location	W1		W2		W3		Best performance
	A	B	A	B	A	B	
pnt_608	0.816	0.814	0.843	0.839	1.063	1.065	Citizen sensor
pnt_610	0.770	0.768	0.882	0.876	0.997	1.000	Citizen sensor
pnt_611	0.644	0.640	1.056	1.050	0.870	0.876	In-situ sensor
pnt_619	0.700	0.705	1.014	1.021	0.888	0.882	In-situ sensor
pnt_620	0.601	0.598	1.093	1.090	0.869	0.873	In-situ sensor
pnt_648	0.589	0.592	1.213	1.215	0.729	0.725	In-situ sensor
pnt_1019	0.580	0.577	1.137	1.134	0.787	0.790	In-situ sensor
pnt_1100	0.823	0.823	0.881	0.887	1.004	1.004	Citizen sensor
pnt_1101	0.984	0.982	0.735	0.744	0.911	0.912	Satellite sensor
pnt_1163	0.910	0.910	0.764	0.769	1.062	1.062	Citizen sensor
pnt_1227	0.953	0.953	0.695	0.687	0.969	0.969	Citizen sensor

From these findings, it was concluded that the satellite sensor is least preferred as a source of rainfall data for July 2015 in this study area as compared to the in-situ and citizen sensors. This is mainly attributed to

the factors discussed in Section 7.1. The RMSE of the weights for each the sources of data using the two approaches were obtained which shows that the weights due to satellite data are the most stable while the weights due to in-situ data had more deviations. Since the average RMSE for satellite, in-situ and citizen sensors using the two approaches was determined to be less than 0.05, our alternative solution provides a reproducible result which supports the success of our method using web services for triple sensor workflow.

	Satellite sensor	In-situ sensor	Citizen sensor
RMSE	0.0089442719	0.0191049732	0.0112694277

In spite of the few differences experienced, successful application of triple sensor approach using the triple collocation method indicates a promising opportunity for improved water monitoring and forecasting. Combining the three sources of data helps eliminate their limitations thereby providing accurate and reliable information for effective water resource management.

8. CONCLUSIONS AND RECOMMENDATIONS

8.1. Conclusions

In this research, we presented two methods for enhancing the shareability and reproducibility of geoprocessing workflows. First, we discussed the current-state-of-art of commonly used geoprocessing workflow management systems (WfMSs) and realized that most of them do not support the composition of workflows from web services. This makes it difficult to share processes for conducting complex distributed geoprocessing tasks using workflows. We agree with the discussion of J. Morales & De By (2009) and Yue et al. (2012) that to support distributed geoprocessing, there is a need for GIS software to distribute their geoprocessing methods as loosely-coupled web services in the context of Service Oriented Architecture (SOA). By doing so, they provide an interoperable computing infrastructure in which geoprocessing workflows can be built and executed with minimal cost to the users. We also noted that the advancement in earth observation and remote-sensing technology, and the rise of Web 2.0 had made the production of massive geospatial data available within a short time. As a result of this, there need to be systems capable of integrating such bulk data in a workflow for distributed computing. Since the processes and data generated by different data providers are disparate, there is a need for standardization to ensure interoperability and accessibility of geoprocessing resources. To support interoperability and accessibility, the OGC has established standards for web services supported by the Web Processing Service (WPS), Web Feature Service (WFS), Web Coverage Service (WCS) and Sensor Observation Service (SOS). We demonstrated that WCS, WFS, and SOS have made it possible to combine satellite data, in-situ measurements and crowdsourced geoinformation in a workflow. WPS, on the other hand, provides an interface in which geoprocessing functions can be shared and accessed by web services. We, therefore, implemented a generic workflow client which make it possible for users to compose workflows by combining geoprocessing functions and geospatial data exposed as web services and execute their workflows in the geoprocessing web without having to install any GIS software.

Secondly, we observed that current WfMSs do not have a standardized interchange format for their workflows. Each GIS software producer comes with their schema to specify their workflows which make sharing and reproduction of workflows impossible across different WfMSs. We noted that there are already standards produced by the Object Management Group and Workflow Management Coalition (WfMC) towards establishing a universal interchange format through BPMN, XML process definition language (XPDL) and Business process definition metamodel (BPDM). However, current GIS WfMSs do not follow these established standards mainly because they were established with a focus to business processes and did not support light-weight exchange formats like JSON which is extensively being adopted by the scientific community. Since most of the current GIS WfMSs support sharing workflows

through JSON, we propose a standard interchange format for sharing workflows based on JSON and also provide a method for transforming workflows from one WfMS to another.

In this research, we were guided by the following research objectives.

1. To investigate existing workflow interchange formats and propose an interoperable standard format for sharing workflows.
2. To devise a method for producing shareable and reproducible workflow.
3. To design and implement a prototype that facilitates the creation and sharing of workflows.
4. To demonstrate the applicability of the prototype in combining crowdsourced geoinformation, in-situ measurements and satellite data for water resource monitoring and forecasting.

We answer the questions related to the objectives in the following ways.

Related to the first objective

i. What are the available tools/software for creating geoprocessing workflows?

In Chapter 3, we discussed available tools for creating geoprocessing workflows. In our research, these tools are put into two categories based on their conformity to established standards. The first category of tools conforms to established standards by organizations for managing workflows like WfMC, OMG, and OGC. These tools were found out to be the BPMN compliant software like JBPMN, Bonita, Camunda modeler, and Yaoqiang BPMN editor. These tools are generic WfMSs and do not have inbuilt geoprocessing functions. They allow the composition of workflows from distributed geoprocessing functions exposed as web processing services (WPS). The second category of tools for creating geoprocessing workflows hardly conform to any standard established by the standardization organization apart from the use of BPMN graphical representations. There are four commonly used tools within the geospatial community under this category. These include ILWIS workflow builder, QGIS processing modeler, ERDAS Imagine Spatial Analyst, and ArcGIS model builder. An example of non-GIS WfMS which do not conform to standards established by this research is KNIME.

ii. Which interchange formats do they use to share their workflows?

The first category of WfMSs uses BPMN schemas to share their workflows. We discussed the schema of the tools in Chapter 3. The schema for the workflow interchange formats of these tools is contained in the BPMN's five XSD files which describe process semantics and its graphical representations. These XSD files include BPMN20.xsd, Semantics.xsd, BPMNDI.xsd, DC.xsd and DC.xsd. BPMN documents have several elements and attributes whose descriptions are well elaborated in Section 3.1. The second category of WfMSs has developed their interchange formats for workflows based on the different schema which raises interoperability concern. ILWIS, QGIS, and ERDAS use JSON file formats. However, all of them have defined their schema for their JSON files. It was not possible to determine the interchange format for ArcGIS generated workflows. In Section 3.2, we discussed the interchange formats for ILWIS and QGIS.

iii. How can a standard interchange format be created to achieve interoperability?

To achieve interoperability, we developed a standard interchange format which reflects at least the commonly used constructs among different software packages. In Section 5.1, we compare the constructs used by current WfMSs and using the observed similarities and differences; we proposed a JSON based standard interchange format in Section 5.2 which can be adopted by software developers to share their workflows. We suggested that a workflow interchange format should have at the top level of the hierarchy four main elements which include an identifier for the workflow, metadata describing the purpose of the workflow, list of operations and connections between the operations. Each operation should have at least one input and output parameters. Metadata providing information about the operation should also be provided. The JSON schema for the interchange format for sharing workflows that were developed is shown in Appendix A.

Related to the second objective

i. What does it take for a workflow to be shared and reproduced?

In Chapter 2, we defined shareability of scientific workflow as the ability to transfer the workflow from one scientist to another or one environment to another in a manner that allows readability and understanding of the workflow that is not necessarily created by the same scientist or in the same environment. We also discussed that reproducibility allows a workflow created for a particular scientific problem to be reused by different users by repetition of steps to produce scientifically similar results.

For a workflow to be shareable and reproducible, it should have the following properties which we discussed in Chapter 2.

- It should be presented in an interoperable interchange format containing a well-defined schema that is universally accepted by developers of WfMSs. A well-defined schema is similar to what we have discussed in Section 5.1.2.
- Third-party resources such as web processing services used in composing the workflows should be available and easily accessible. The developers of GIS software should avail their geoprocessing functions as web services.
- There should be sufficient input data to reproduce the workflow. Whenever a mandatory data required by a process cannot be found, the execution of the entire workflow fails. Geospatial data varies by scale, resolution and coordinate system. When incompatible data are used together, they introduce errors which affect the reproducibility of the workflow.
- There should be sufficient metadata information for the workflow. The metadata should provide descriptive information about the processes, input and output data, and connections between processes.

ii. How can a workflow be composed of distributed geospatial web services?

In Chapter 4, we discussed the method by which a workflow can be composed of geospatial web services. First, we addressed the issue of composability of workflows by looking at different levels of composability. After that, we discussed how web services could be used to compose workflows by looking at the OGC standards for sharing and accessing data and processes which include WPS, WFS, WCS, and SOS. Apart from the OGC web services, we found out that there exist other RESTful web services which do not conform to any universal standards but can also provide geoprocessing functions. We found out that the composition of workflow from web services require a generic workflow client that allows users to drag and drop services and connect the processing services visually using BPMN's graphics.

We also identified that a composed workflow would require a workflow engine in which its execution can occur. Since current WfMSs have their limitations which we observed in Chapter 3, we proposed a method which if implemented can be used to chain processing services using one of the OGC process chaining techniques. Once we have chained the process, we can execute them using the workflow engine and relay the result to the user.

iii. *How can a workflow be shared across different geoprocessing tools/software?*

To support sharing of geoprocessing workflows, we developed a standard interchange schema based on JSON format which can be used in specifying a workflow. This JSON schema acts as an intermediary between different interchange formats and allows transformation of workflow from one WfMSs to another using the information from their interchange formats. We also noted that the transformation of workflows between different WfMSs requires knowledge of corresponding processes in the target software. We found out that one way which has been proved to help in the discovery of geoprocesses is through the use of semantic web technology and ontologies. Though this was beyond the scope of this study, we consider it as an instrumental technique that if integrated into our method can help in sharing workflows across different geoprocessing tools. However, an alternative solution was provided that can perform a simple search from a database of processes and return a corresponding process based on a search keyword.

Related to the third objective

i. *How can the prototype system be developed?*

In Section 6.2, we discussed the implementation of the prototype system to support the shareability and reproducibility of workflows. For the implementation of the workflow client, we require the ExtJS JavaScript framework, D3 JS and OpenLayers while the workflow engine requires Python programming language. A single-web page application was developed with all the functionalities that support creation, sharing and reproduction of workflows. We used BPMN diagrams for visual composition of the workflows where the nodes represent a web processing service, and the edges represent connections between the processing services. The visual representation of the workflow is automatically translated to a lightweight data exchange format in JSON which can be sent to the workflow engine for execution. We implement several

functions for the workflow engine which support process chaining, workflow execution and transformation of workflow from one WfMS to another.

ii. *What are the requirements and procedure for setting up the system?*

The required components for setting up the system are outlined in Section 6.1. These include:

- Apache HTTP web server version 2.4.
- Python 3.6
- Apache Tomcat 9
- GeoServer
- PostgreSQL
- ILWIS 3 and 4
- QGIS 2.18

The installation and configuration instructions that were used for these software packages are described in Appendix F. However, in case these instructions are not sufficient, the official installation instructions can be obtained online in the vendor's websites. The procedure for setting up the system after installing the above software packages are as follows.

- Download the web files from GitHub¹⁹.
- Extract the files to the root folder of Apache HTTP server.
- Start your Apache HTTP server if it wasn't running.
- Go to the preferred web browser and browse to the location of the index file.

iii. *What are the limitations to this system and the problems that can be encountered?*

- The system only works with Geotiff and GeoJSON data formats. The use of web services which implements other data formats can lead to errors. The available download options for data is also supported only for Geotiff and GeoJSON file formats.
- Since the system is based on distributed processing, availability of processes exposed as web services is mandatory. Whenever a process has been redefined by the service provider and is not updated in the workflow, this can affect the reproducibility of the workflow.
- Availability of the data must be sustained to ensure reproducibility of the workflow. The system fails if the path to the data is changed.
- Consistent internet connection is required to sustain communication between the client and the workflow engine.
- Failure to adhere to the specification defined in the JSON schema for workflow sharing can lead to errors. The required input parameters for processing services must be supplied.

Related to the fourth objective

¹⁹ <https://github.com/robertohuru/WorkflowApp>

i. What are the potential characteristics of crowdsourced geoinformation, satellite and in-situ data that affects their combination?

We discussed in Section 7.1 five essential characteristics that affect the combination of crowdsourced geoinformation with satellite and in-situ data.

Data variable: The chosen data variable should be the same for all the sources to ensure an effective combination. For instance, it is not possible to combine rainfall and temperature using the triple collocation method.

Data representation format (data type): A unique representation format must be used to describe the data in each of the three sources. For example, combining a Boolean and nominal variable will not yield a positive result since they are incomparable.

Temporal resolution: The period in which the sampling was carried out must be the same for all the three sources of data. For the satellite data, the sampling period can be affected by the temporal resolution of the satellite. Most in-situ stations reporting is done regularly which can occur at an hourly or daily basis. This is different from citizen observations which may not be done at regular intervals. Matching of the periods for three data sources is necessary for effective comparison and validation.

Spatial resolution: The observed data from the three sources must be occupying the same geographical space to be able to align them. The geographical space in the triple sensor workflow is determined by defining a region of interest. Since the satellite sensor covers a large extent, clipping and resampling can be used to obtain data for the region of interest. The low density of in-situ stations affects the combination of these data sources and can provide unreliable result after interpolation. The density of crowdsourced geoinformation does not affect the combination since it is used as the reference data to determine the region of interest.

Coordinate system: For effective alignment, the data must be in the same coordinate system. In case the data are of the different coordinate system, then they must be projected or transformed into one coordinate system. However, if this is not done the triple sensor approach fails.

Data quality: The success of the combination of crowdsourced geoinformation with satellite and in-situ data using triple-sensor approach lies significantly in the quality of the observed data. Low quality of data which is mostly attributed to citizen-based observations can provide an inaccurate result.

ii. How can specific operations be integrated to combine crowdsourced geoinformation, satellite, and in-situ data?

In Section 7.3.3, we discussed specific operations which are required to successfully implement the triple sensor approach for combining crowdsourced geoinformation, satellite, and in-situ data. Since there are no exposed web services for these operations, we implement web processing services for these operations using the specifications for OGC WPS. Implementation of WCS for satellite data and SOS for crowdsourced geoinformation and in-situ data was carried out. These

web services were used in the workflow client to compose a shareable triple collocation workflow for combining data from the three sources and evaluating their performance for specific locations.

iii. *What is the added value of the method to shareability and reproducibility of workflow for integration of crowdsourced geoinformation, satellite, and in-situ data?*

The approach followed by Mannaerts et al. (2018) in combining the crowdsourced geoinformation, satellite and in-situ data was entirely based on an implementation using a desktop GIS tool, ILWIS, which required users to install the software and execute the workflow with a pre-processed data. Since the ILWIS version they used runs on a Windows-driven operating system, their approach is not interoperable as it prevents the reproduction of their method in other operating systems. Their approach also did not provide a shareable workflow specification which can be exported to other systems. The use of pre-processed in-situ data eliminated other vital processes in the execution chain which affects reproduction of their method. In our method, we proposed and demonstrated two approaches for enhancing the shareability and reproducibility of workflows. One of them being the composition of workflow from web services. This approach makes it possible to incorporate crowdsourced geoinformation and in-situ data using the OGC SOS specification in a workflow. The implementation of the web processing services ensures interoperability which is very important for enhancing shareability and reproducibility of workflows. The second approach involves the use of a standard workflow interchange schema which acts a link to transform from one workflow interchange format to another. This makes it possible for us to create a workflow and reuse it in different WfMSs without the need to recreate it.

8.2. Limitations

The following limitations were encountered during the progress of this research.

- Insufficient provenance information for QGIS workflows makes it difficult to reproduce workflows. There is little metadata information provided for the non-spatial input parameters.
- It was not possible to determine the interchange schema for ArcGIS model builder since the workflow is stored in a format which can only be read inside the ArcGIS environment.
- Chaining of processes using the OGC WPS is not possible when using different WPS servers. The HTTP POST request made for a WPS assumes all the processes are hosted in the same server in which the request is sent.
- Insufficient geoprocessing web services available since current GIS software do not expose their geoprocessing functions as web services.
- Different implementation requirements make the discovery of similar geoprocesses in corresponding GIS software difficult. This is because the geoprocessing functions implemented in each GIS software have different input requirements.

- Our research focused on two levels of composability which included structural composability and static syntactic composability. However, we cannot only rely on this to identify all error for effective workflow composition. Diniz (2016), demonstrated the use of semantic composability, however, since we propose a different schema for our workflows, it was not possible to use his methods to check for semantic composability of our workflows.
- Despite getting similar results with Mannaerts et al. (2018) when using our method for demonstrating the triple sensor workflow as illustrated in Section 7.4, we were not able to provide a quantitative measure for shareability and reproducibility of the workflow. There does not exist such a justifiable method which has been used to measure shareability and reproducibility of scientific workflows quantitatively. However, we found out that research by Zhao et al. (2012) addressed factors that can affect the reproducibility of scientific workflows.
- Reproduction of ILWIS workflows from JSON file formats into the ILWIS WfMS is not supported. This made it difficult to test the reproducibility of workflows in ILWIS. ILWIS currently support only exporting of workflows from their internal format to the interoperable JSON format.

8.3. Suggestions for OGC Standards

- The OGC process chaining should extend to compliant and non-compliant OGC WPS RESTful bindings. The current implementations of process chaining only support compliant OGC WPS SOAP bindings. In the case of OGC WPS RESTful services, the identifier keyword in the body of HTTP POST request should be optional.
- The body of Execute request of WPS should contain a keyword for the URI of the WPS server for every WPS process in the chain. This should be implemented in such a manner that each process in the chain is independent of the WPS server used in the HTTP POST request to enable service calls to different WPS servers. Currently, the executing body of a process chain assumes that all the processes are provided by one WPS server making it impossible to chain WPS of different WPS servers within a WPS execute the operation.
- Introduce JSON-based RESTful bindings for SOS. Currently, only WFS support RESTful bindings.
- The default data format for WPS execution result should be a GeoTIFF or GeoJSON for WCS and WFS respectively. This introduces a uniform format thereby reducing complexity for the orchestrating engine when passing data between different processes in a workflow.

8.4. Suggestion for GIS Software Developers

- Standard workflow exchange schema: We propose a standard workflow exchange schema for the developers of GIS WfMSs. Since most of the GIS WfMSs use JSON file format, our proposed

workflow exchange format which is also JSON based provides a universal means in which interoperability can be achieved for sharing their workflows.

- Web Service Enablement: We recommend to GIS software developers to implement their workflow editors to allow composition of workflow from web services using WPS, WCS, and WFS. Currently, none of the GIS WfMSs support composition of workflows from web services. Though most of their WfMSs are desktop based, we believe that web services offer a more reliable environment for executing complex processes and bulk data.
- ILWIS Developers: (i) Capture the screen position of the processes (nodes) in the workflow exchange format to facilitate visual recreation of the workflow; (ii) Consider removing irrelevant keywords in the schema of the workflow exchange format which can increase the complexity of the workflow. For instance, the keywords like *change*, *show* and *local* in inputs and outputs, and *final* in operation metadata are not necessary; (iii) Implement a method to reproduce ILWIS generated workflows from JSON format. Currently, ILWIS only support exporting workflow to a JSON file but importing the workflow back to ILWIS is not possible.
- QGIS Developers: (i) Provide a consistent schema for the workflows which captures enough metadata for the workflow. For instance, currently, the schema only provides metadata information for spatial inputs while non-spatial data are assigned as values with no description provided.

8.5. Recommendations for Future Work

- Semantic Web and Ontology: Future work should employ the use of Semantic web technology and Ontology to discover geoprocessing functions of GIS software and web services based on their descriptions. This research can extend on the previous research Ubels (2018) on the use of semantic web technology to facilitate discovery of geoprocessing functions.
- XML to JSON transformations: Kechagioglou & Lemmens (2018) has been researching on sharing geoprocessing workflows of ILWIS with BPMN. Our approaches differ slightly in that we focus on using JSON while they use XML and BPMN schema. The future work can find out how to perform transformation between XML and JSON schema. Also, workflow engines for executing BPMN workflows should be addressed in future work.
- Verification of composability of geoprocessing workflows: There was not enough time to perform complete verification of workflow composability as per our discussion in Section 4.1. Our research focused on structural and static syntactic composability which cannot provide the much-required verification of composability. Application of dynamic syntactic composability, semantic composability, and qualitative composability is needed to ensure higher reliability during workflow execution.

- A measure of shareability and reproducibility: Even though this research provided a method for enhancing shareability and reproducibility of workflows, this has not been proven since we could not obtain a quantifiable measure for shareability and reproducibility. A method for determining shareability and reproducibility is needed to improve the applicability of our method.
- ERDAS and ArcMap extension: Future work should provide an extension for ERDAS and ArcMap which are also commonly used GIS WfMSs. Although ArcMap model builder support sharing of workflows using Python files, much is still needed to transform ArcMap workflows to shareable file formats like JSON.

LIST OF REFERENCES

- Amsden, J., Frank, J., Gardner, T., Irassar, P., Iyengar, S., Johnston, S., & White Stephen. (2004). *Business Process Definition Metamodel*. Retrieved from <https://www.omg.org/bpmn/Documents/BPDM/OMG-BPD-2004-01-12-Revision.pdf>
- Assumpcao, T. H., Popescu, I., Jonoski, A., & Solomatine, D. P. (2018). Citizen observations contributing to flood modelling: Opportunities and challenges. *Hydrology and Earth System Sciences*, 22(2), 1473–1489. <https://doi.org/10.5194/hess-22-1473-2018>
- Banati, A., Kacsuk, P., & Kozlovsky, M. (2015). Four level provenance support to achieve portable reproducibility of scientific workflows. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 241–244). IEEE. <https://doi.org/10.1109/MIPRO.2015.7160272>
- Banati, A., Kacsuk, P., & Kozlovsky, M. (2016). Evaluating the reproducibility cost of the scientific workflows. In *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)* (pp. 187–190). IEEE. <https://doi.org/10.1109/SACI.2016.7507367>
- Barga, R., & Gannon, D. (2007). Scientific versus Business Workflows. In *Workflows for e-Science* (pp. 9–16). London: Springer London. https://doi.org/10.1007/978-1-84628-757-2_2
- Bechhofer, S., Buchan, I., De Roure, D., Missier, P., Ainsworth, J., Bhagat, J., ... Goble, C. (2013). Why linked data is not enough for scientists. *Future Generation Computer Systems*, 29(2), 599–611. <https://doi.org/10.1016/J.FUTURE.2011.08.004>
- Belhajjame, K., Vargas-Solar, G., & Collet, C. (2002). A flexible workflow model for process-oriented applications. In *Proceedings of the Second International Conference on Web Information Systems Engineering* (pp. 72–80). Saint-Martin d’Heres, France: IEEE Comput. Soc. <https://doi.org/10.1109/WISE.2001.996468>
- Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., ... Lemmens, R. (2011). New Generation Sensor Web Enablement. *Sensors*, 11, 2652–2699. <https://doi.org/10.3390/s110302652>
- Burattin, A. (2015). Introduction to Business Processes, BPM, and BPM Systems (pp. 11–21). Springer, Cham. https://doi.org/10.1007/978-3-319-17482-2_2
- Chu, X., Kobialka, T., Durnota, B., & Buyya, R. (2006). Open sensor web architecture: Core services. In *Proceedings - 4th International Conference on Intelligent Sensing and Information Processing, ICISIP 2006* (pp. 98–103). IEEE. <https://doi.org/10.1109/ICISIP.2006.4286069>
- Clark, J. M., Schaeffer, B. A., Darling, J. A., Urquhart, E. A., Johnston, J. M., Ignatius, A. R., ... Stumpf, R. P. (2017). Satellite monitoring of cyanobacterial harmful algal bloom frequency in recreational waters and drinking water sources. *Ecological Indicators*, 80, 84–95. <https://doi.org/10.1016/J.ECOLIND.2017.04.046>
- Curcin, V., & Ghanem, M. (2008). Scientific workflow systems - can one size fit all? In *2008 Cairo International Biomedical Engineering Conference* (pp. 1–9). IEEE.

<https://doi.org/10.1109/CIBEC.2008.4786077>

- Decker, M., Che, H., Oberweis, A., Stürzel, P., & Vogel, M. (2010). Modelling Mobile Workflows with BPMN. In *2010 Ninth International Conference on Mobile Business and 2010 Ninth Global Mobility Roundtable (ICMB-GMR)* (pp. 272–279). IEEE. <https://doi.org/10.1109/ICMB-GMR.2010.12>
- Dhib, S., Mannaerts, C. M., Bargaoui, Z., Retsios, V., & Maathuis, B. H. P. (2017). Evaluating the MSG satellite Multi-Sensor Precipitation Estimate for extreme rainfall monitoring over northern Tunisia. *Weather and Climate Extremes*, 16, 14–22. <https://doi.org/10.1016/J.WACE.2017.03.002>
- Diniz, F. D. E. C. (2016). Composition of Semantically Enabled Geospatial Web Services. *MSC Theses*, 142.
- Garcia, L., Rodriguez, D., Wijnen, M., & Pakulski, I. (2016). *Earth Observation for Water Resources Management*. Washington DC. Retrieved from <https://openknowledge.worldbank.org/bitstream/handle/10986/22952/9781464804755.pdf?sequence=3&isAllowed=y>
- Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., ... Myers, J. (2007). Examining the Challenges of Scientific Workflows. *Computer*, 40(12), 24–32. <https://doi.org/10.1109/MC.2007.421>
- Gonçalves, P. (2017). *OGC Testbed-13: Application Deployment and Execution Service ER*. Retrieved from http://docs.opengeospatial.org/per/17-024.html#OGC_16_035
- Goodchild, M. F. (2007). Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4), 211–221. <https://doi.org/10.1007/s10708-007-9111-y>
- Hollingsworth, D. (1995). *The Workflow Reference Model. Workflow Management Coalition* (Vol. 59). Hampshire, UK. <https://doi.org/citeulike-article-id:1378584>
- Juhnke, E., Dornemann, T., Kirch, S., Seiler, D., & Freisleben, B. (2010). SimpleBPEL: Simplified Modelling of BPEL Workflows for Scientific End Users. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications* (pp. 137–140). IEEE. <https://doi.org/10.1109/SEAA.2010.32>
- Kechagioglou, X., & Lemmens, R. (2018). Sharing geoprocessing workflows with Business Process Model and Notation (BPMN). *Research Paper*, 2–7.
- Ko, R. K. L., Lee, S. S. G., & Lee, E. W. (2009). Business process management (BPM) standards: a survey. *Business Process Management Journal*, 15(5), 744–791. <https://doi.org/10.1108/14637150910987937>
- Lemmens, R., Schouwenburg, M., Retsios, B., Mannaerts, C., & Ronzhin, S. (2018). Using ILWIS Software for teaching Core Operations in Earth Observation, 2–4.
- Lemmens, R., Toxopeus, B., Boerboom, L., Schouwenburg, M., Retsios, B., Nieuwenhuis, W., & Mannaerts, C. (2018). Implementation of a comprehensive and effective geoprocessing workflow environment. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 42(4W8), 123–127. <https://doi.org/10.5194/isprs-archives-XLII-4-W8-123-2018>
- Leroux, D. J., Kerr, Y. H., Richaume, P., & Berthelot, B. (2011). Estimating SMOS error structure using

- triple collocation. In *International Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 24–27). IEEE.
<https://doi.org/10.1109/IGARSS.2011.6048888>
- Li, X., McColl, K. A., Lyu, H., Xu, X., Derksen, C., Lu, H., & Entekhabi, D. (2017). Validation of the SMAP freeze/thaw product using categorical triple collocation. In *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 1596–1598). IEEE.
<https://doi.org/10.1109/IGARSS.2017.8127277>
- Mannaerts, C., Maathuis, B., Wehn, U., Gerrets, T., Riedstra, H., Becht, R., ... Kwast, H. V. D. (2017a). Deliverable Title Constraints and Opportunities for Water Resources Monitoring & Forecasting using the Triple Sensor approach Status Final Related Work Package WP4 Deliverable lead ITC Versions and Contribution History. Retrieved from <https://afrialliance.org/wp-content/uploads/sites/33/2018/02/D4.4-Constraints-Opportunities-MF-Triplesensor-final.pdf>
- Mannaerts, C., Maathuis, B., Wehn, U., Gerrets, T., Riedstra, H., Becht, R., ... Kwast, H. V. D. (2017b). *Deliverable Title Constraints and Opportunities for Water Resources Monitoring & Forecasting using the Triple Sensor approach Status Final Related Work Package WP4 Deliverable lead ITC Versions and Contribution History*. Retrieved from <https://afrialliance.org/wp-content/uploads/sites/33/2018/02/D4.4-Constraints-Opportunities-MF-Triplesensor-final.pdf>
- Mannaerts, C., Retsios, B., & Maathuis, B. (2018). *Deliverable D4 . 6 (report supplement) Description of the Demonstration Package for Monitoring and Fore- casting Water & Climate using a Triple Sensor approach* (Vol. 6).
- McColl, K. A., Roy, A., Derksen, C., Konings, A. G., Alemohammed, S. H., & Entekhabi, D. (2016). Triple collocation for binary and categorical variables: Application to validating landscape freeze/thaw retrievals. *Remote Sensing of Environment*, 176, 31–42.
<https://doi.org/10.1016/j.rse.2016.01.010>
- McColl, K. A., Vogelzang, J., Konings, A. G., Entekhabi, D., Piles, M., & Stoffelen, A. (2014). Extended triple collocation: Estimating errors and correlation coefficients with respect to an unknown target. *Geophysical Research Letters*, 41(17), 6229–6236. <https://doi.org/10.1002/2014GL061322>
- Medjahed, B., & Bouguettaya, A. (2005). A multilevel composability model for semantic Web services. *IEEE Transactions on Knowledge and Data Engineering*, 17(7), 954–968.
<https://doi.org/10.1109/TKDE.2005.101>
- Meek, S., Jackson, M., & Leibovici, D. G. (2016). A BPMN solution for chaining OGC services to quality assure location-based crowdsourced data. *Computers and Geosciences*, 87, 76–83.
<https://doi.org/10.1016/j.cageo.2015.12.003>
- Mendling, J., Mendling, J., & Neumann, G. (2004). G.: A Comparison of XML Interchange Formats for Business Process Modelling. IN: *PROCEEDINGS OF EMISA 2004 - INFORMATION SYSTEMS IN E-BUSINESS AND E-GOVERNMENT*. LNI, 56, 129--140. Retrieved from <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.3243>
- Mendling, J., & Weidlich, M. (2012). Lecture Notes in Business Information Processing: Preface. *Lecture*

- Notes in Business Information Processing*, 125 LNBIP. <https://doi.org/10.1007/978-3-642-33155-8>
- Morales, J., & De By, R. A. (2009). Design templates for real-time geo-processing workflows. In *Digital earth in action*. Beijing, China: International Society for Digital Earth (ISDE). Retrieved from https://webapps.itc.utwente.nl/library/2009/conf/morales_des.pdf
- Morales, J. M. (2004). *Model-driven Design of Geo-information Services*. University of Twente, Enschede, The Netherlands. Retrieved from https://webapps.itc.utwente.nl/librarywww/papers_2004/phd/morales.pdf
- Nurseitov, N., Paulson, M., Reynolds, R., & Izurieta, C. (2009). Comparison of JSON and XML Data Interchange Formats: A Case Study. In *Proceedings of the ISCA 22nd International Conference on Computer Applications in Industry and Engineering*. San Francisco, California, USA. Retrieved from <https://www.cs.montana.edu/izurieta/pubs/IzurietaCAINE2009.pdf>
- OGC. (2012). OGC Sensor Observation Service. *OGC Implementation Standard*, 163. <https://doi.org/OGC-12-006>
- OMG. (2003). MDA Guide v1.0.1. Object Management Group. Retrieved from https://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf
- OMG. (2011). Notation (BPMN) Version 2.0. *OMG Specification, Object Management Group*, (January). [https://doi.org/10.1016/S0020-1693\(97\)05933-1](https://doi.org/10.1016/S0020-1693(97)05933-1)
- Open Geospatial Consortium. (2012). OGC Web Processing Service 1.0. *Agenda*, 1–18.
- Pierdicca, N., Anniballe, R., Noto, F., Bignami, C., Chini, M., Martinelli, A., & Mannella, A. (2018). Triple collocation to assess classification accuracy without a ground truth in case of earthquake damage assessment. *IEEE Transactions on Geoscience and Remote Sensing*, 56(1), 485–496. <https://doi.org/10.1109/TGRS.2017.2750770>
- Pross, B., & Christoph, S. (2018). OGC Testbed-13: Workflows ER. Retrieved September 18, 2018, from <http://docs.opengeospatial.org/per/17-029r1.html>
- Rosser, J. F., Jackson, M., & Leibovici, D. G. (2018). Full Meta Object profiling for flexible geoprocessing workflows. *Transactions in GIS*, 22(5), 1221–1237. <https://doi.org/10.1111/tgis.12460>
- Rouached, M., Baccar, S., & Abid, M. (2012). RESTful sensor web enablement services for wireless sensor networks. In *Proceedings - 2012 IEEE 8th World Congress on Services, SERVICES 2012* (pp. 65–72). IEEE. <https://doi.org/10.1109/SERVICES.2012.48>
- Schäffer, B., & Foerster, T. (2008). A client for distributed geo-processing and workflow design. *Journal of Location Based Services*, 2(3), 194–210. <https://doi.org/10.1080/17489720802558491>
- Scheider, S., & Ballatore, A. (2018). Semantic typing of linked geoprocessing workflows. *International Journal of Digital Earth*, 11(1), 113–138. <https://doi.org/10.1080/17538947.2017.1305457>
- Schmidt, M. T. (1999). The evolution of workflow standards. *IEEE Concurrency*, 7(3), 44–52. <https://doi.org/10.1109/4434.788778>

- Simonis, I., De Lathouwer, B., & Taylor, T. (2016). Sensor Web Enablement (SWE) for citizen science. In *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)* (pp. 3618–3620). IEEE.
<https://doi.org/10.1109/IGARSS.2016.7729937>
- Singh, A. (2016). Understanding Separation Of Concern in ASP.NET MVC. Retrieved February 19, 2019, from <https://www.c-sharpcorner.com/UploadFile/56fb14/understanding-separation-of-concern-and-Asp-Net-mvc/>
- Sonntag, M., Karastoyanova, D., & Deelman, E. (2010). Bridging the gap between business and scientific workflows: Humans in the loop of scientific workflows. *Proceedings - 2010 6th IEEE International Conference on e-Science, EScience 2010*, (December), 206–213. <https://doi.org/10.1109/eScience.2010.12>
- Taylor, I. J., Deelman, E., Gannon, D., & Shields, M. S. (2007). Workflows for e-Science: Scientific Workflows for Grids. *Workflows for E-Science: Scientific Workflows for Grids*, 1–523.
<https://doi.org/10.1007/978-1-84628-757-2>
- Ubels, S. (2018). Understanding abstract geo-information workflows and converting them to executable workflows using Semantic Web technologies MSc Thesis Sam Ubels Department of Geoinformation Processing Faculty of Geoinformation Science and Earth Observation University of.
- Werling, M. (2008). OGC® OWS-5 GeoProcessing Workflow Architecture Engineering Report. *Engineering*.
- Yue, P., Foerster, T., & Zhao, P. (2012). The Geoprocessing Web. *Computers & Geosciences*, 47, 3–12.
<https://doi.org/10.1016/J.CAGEO.2012.04.021>
- Yue, P., Sun, Z., Gong, J., Di, L., & Lu, X. (2011). A provenance framework for Web geoprocessing workflows. In *IEEE International Geoscience and Remote Sensing Symposium* (pp. 3811–3814). IEEE.
<https://doi.org/10.1109/IGARSS.2011.6050061>
- Zhao, J., Gomez-Perez, J. M., Belhajjame, K., Klyne, G., Garcia-Cuesta, E., Garrido, A., ... Goble, C. (2012). Why workflows break — Understanding and combating decay in Taverna workflows. In *2012 IEEE 8th International Conference on E-Science* (pp. 1–9). IEEE.
<https://doi.org/10.1109/eScience.2012.6404482>

Appendix

Appendix A: JSON Schema for Workflow Sharing

```

1. {
2.   "$schema": "http://json-schema.org/draft-04/schema#",
3.   "type": "object",
4.   "properties": {
5.     "workflows": {
6.       "type": "array",
7.       "items": [
8.         {
9.           "type": "object",
10.          "properties": {
11.            "id": {
12.              "type": "integer"
13.            },
14.            "metadata": {
15.              "type": "object",
16.              "properties": {
17.                "longname": {
18.                  "type": "string"
19.                }
20.              },
21.              "required": [
22.                "longname"
23.              ]
24.            },
25.            "operations": {
26.              "type": "array",
27.              "items": [
28.                {
29.                  "type": "object",
30.                  "properties": {
31.                    "id": {
32.                      "type": "integer"
33.                    },
34.                    "metadata": {
35.                      "type": "object",
36.                      "properties": {
37.                        "longname": {
38.                          "type": "string"
39.                        },
40.                        "label": {
41.                          "type": "string"
42.                        },
43.                        "url": {
44.                          "type": "string"
45.                        },
46.                        "resource": {
47.                          "type": "string"
48.                        },
49.                        "description": {
50.                          "type": "string"
51.                        },
52.                        "inputparametercount": {
53.                          "type": "integer"
54.                        },
55.                        "outputparametercount": {
56.                          "type": "integer"
57.                        },
58.                        "position": {

```

```

59.         "type": "array",
60.         "items": [
61.             {
62.                 "type": "integer"
63.             },
64.             {
65.                 "type": "integer"
66.             }
67.         ]
68.     },
69.     "required": [
70.         "longname",
71.         "label",
72.         "url",
73.         "resource",
74.         "description",
75.         "inputparametercount",
76.         "outputparametercount",
77.         "position"
78.     ]
79. },
80. "inputs": {
81.     "type": "array",
82.     "items": [
83.         {
84.             "type": "object",
85.             "properties": {
86.                 "id": {
87.                     "type": "integer"
88.                 },
89.                 "identifier": {
90.                     "type": "string"
91.                 },
92.                 "name": {
93.                     "type": "string"
94.                 },
95.                 "type": {
96.                     "type": "string"
97.                 },
98.                 "description": {
99.                     "type": "string"
100.                 },
101.                 "optional": {
102.                     "type": "boolean"
103.                 },
104.                 "url": {
105.                     "type": "string"
106.                 },
107.                 "value": {
108.                     "type": "string"
109.                 }
110.             },
111.             "required": [
112.                 "id",
113.                 "identifier",
114.                 "name",
115.                 "type",
116.                 "description",
117.                 "optional",
118.                 "url",
119.                 "value"
120.             ]
121.         }
122.     ]

```



```

123.         ]
124.     },
125.     "outputs": {
126.         "type": "array",
127.         "items": [
128.             {
129.                 "type": "object",
130.                 "properties": {
131.                     "id": {
132.                         "type": "integer"
133.                     },
134.                     "identifier": {
135.                         "type": "string"
136.                     },
137.                     "name": {
138.                         "type": "string"
139.                     },
140.                     "value": {
141.                         "type": "string"
142.                     },
143.                     "description": {
144.                         "type": "string"
145.                     },
146.                     "type": {
147.                         "type": "string"
148.                     }
149.                 },
150.                 "required": [
151.                     "id",
152.                     "identifier",
153.                     "name",
154.                     "value",
155.                     "description",
156.                     "type"
157.                 ]
158.             }
159.         ]
160.     },
161.     "required": [
162.         "id",
163.         "metadata",
164.         "inputs",
165.         "outputs"
166.     ]
167. },
168. {
169.     "type": "object",
170.     "properties": {
171.         "id": {
172.             "type": "integer"
173.         },
174.         "metadata": {
175.             "type": "object",
176.             "properties": {
177.                 "longname": {
178.                     "type": "string"
179.                 },
180.                 "label": {
181.                     "type": "string"
182.                 },
183.                 "url": {
184.                     "type": "string"
185.                 }
186.             },

```

```

187.         "resource": {
188.             "type": "string"
189.         },
190.         "description": {
191.             "type": "string"
192.         },
193.         "inputparametercount": {
194.             "type": "integer"
195.         },
196.         "outputparametercount": {
197.             "type": "integer"
198.         },
199.         "position": {
200.             "type": "array",
201.             "items": [
202.                 {
203.                     "type": "integer"
204.                 },
205.                 {
206.                     "type": "integer"
207.                 }
208.             ]
209.         },
210.     },
211.     "required": [
212.         "longname",
213.         "label",
214.         "url",
215.         "resource",
216.         "description",
217.         "inputparametercount",
218.         "outputparametercount",
219.         "position"
220.     ],
221.     "inputs": {
222.         "type": "array",
223.         "items": [
224.             {
225.                 "type": "object",
226.                 "properties": {
227.                     "id": {
228.                         "type": "integer"
229.                     },
230.                     "identifier": {
231.                         "type": "string"
232.                     },
233.                     "name": {
234.                         "type": "string"
235.                     },
236.                     "type": {
237.                         "type": "string"
238.                     },
239.                     "description": {
240.                         "type": "string"
241.                     },
242.                     "optional": {
243.                         "type": "boolean"
244.                     },
245.                     "url": {
246.                         "type": "string"
247.                     },
248.                     "value": {
249.                         "type": "string"
250.                     }

```

```

251.         }
252.     },
253.     "required": [
254.         "id",
255.         "identifier",
256.         "name",
257.         "type",
258.         "description",
259.         "optional",
260.         "url",
261.         "value"
262.     ]
263. },
264. {
265.     "type": "object",
266.     "properties": {
267.         "id": {
268.             "type": "integer"
269.         },
270.         "identifier": {
271.             "type": "string"
272.         },
273.         "name": {
274.             "type": "string"
275.         },
276.         "type": {
277.             "type": "string"
278.         },
279.         "description": {
280.             "type": "string"
281.         },
282.         "optional": {
283.             "type": "boolean"
284.         },
285.         "url": {
286.             "type": "string"
287.         },
288.         "value": {
289.             "type": "string"
290.         }
291.     },
292.     "required": [
293.         "id",
294.         "identifier",
295.         "name",
296.         "type",
297.         "description",
298.         "optional",
299.         "url",
300.         "value"
301.     ]
302. },
303. {
304.     "type": "object",
305.     "properties": {
306.         "id": {
307.             "type": "integer"
308.         },
309.         "identifier": {
310.             "type": "string"
311.         },
312.         "name": {
313.             "type": "string"
314.         },

```

```

315.         "type": {
316.             "type": "string"
317.         },
318.         "description": {
319.             "type": "string"
320.         },
321.         "optional": {
322.             "type": "boolean"
323.         },
324.         "url": {
325.             "type": "string"
326.         },
327.         "value": {
328.             "type": "string"
329.         }
330.     },
331.     "required": [
332.         "id",
333.         "identifier",
334.         "name",
335.         "type",
336.         "description",
337.         "optional",
338.         "url",
339.         "value"
340.     ]
341. }
342. ]
343. },
344. "outputs": {
345.     "type": "array",
346.     "items": [
347.         {
348.             "type": "object",
349.             "properties": {
350.                 "id": {
351.                     "type": "integer"
352.                 },
353.                 "identifier": {
354.                     "type": "string"
355.                 },
356.                 "name": {
357.                     "type": "string"
358.                 },
359.                 "value": {
360.                     "type": "string"
361.                 },
362.                 "description": {
363.                     "type": "string"
364.                 },
365.                 "type": {
366.                     "type": "string"
367.                 }
368.             },
369.             "required": [
370.                 "id",
371.                 "identifier",
372.                 "name",
373.                 "value",
374.                 "description",
375.                 "type"
376.             ]
377.         }
378.     ]

```

```

379.         }
380.     },
381.     "required": [
382.         "id",
383.         "metadata",
384.         "inputs",
385.         "outputs"
386.     ]
387. }
388. ]
389. },
390. "connections": {
391.     "type": "array",
392.     "items": [
393.         {
394.             "type": "object",
395.             "properties": {
396.                 "fromOperationID": {
397.                     "type": "integer"
398.                 },
399.                 "toOperationID": {
400.                     "type": "integer"
401.                 },
402.                 "fromParameterID": {
403.                     "type": "integer"
404.                 },
405.                 "toParameterID": {
406.                     "type": "integer"
407.                 }
408.             },
409.             "required": [
410.                 "fromOperationID",
411.                 "toOperationID",
412.                 "fromParameterID",
413.                 "toParameterID"
414.             ]
415.         }
416.     ]
417. },
418. },
419. "required": [
420.     "id",
421.     "metadata",
422.     "operations",
423.     "connections"
424. ]
425. }
426. ]
427. }
428. },
429. "required": [
430.     "workflows"
431. ]
432. }

```

Appendix B: JSON Representation for Triple Sensor Water Accounting Workflow

```

1.  {
2.    "workflows": [
3.      {
4.        "id": 1,

```

```

5.     "metadata": {
6.         "longname": "Subworkflow"
7.     },
8.     "operations": [
9.         {
10.            "id": 0,
11.            "metadata": {
12.                "longname": "i3:Moving Average",
13.                "label": "moving_wps_average",
14.                "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
15.                "resource": "WPS",
16.                "description": "The Moving average operation is a point interpolation which requires a
point map as input and returns a raster map as output. To the output pixels, weighted averaged
point values are assigned. The weight factors for the points are calculated by a user-
specified weight function. The weight function ensures that points close to an output pixel obta
in larger weights than points which are farther away. Furthermore, the weight functions are imp
lemented in such a way that points which are farther away from an output pixel than a user-
defined limiting distance obtain weight zero. When interpolating point values, it is for time effici
ency reasons, strongly advised to choose a rather large pixel size for the output map. Further in
terpolation on the raster map values can be performed using the Densify operation or the Resa
mple operation.",
17.                "inputparametercount": 6,
18.                "outputparametercount": 1,
19.                "position": [
20.                    509,
21.                    30
22.                ]
23.            },
24.            "inputs": [
25.                {
26.                    "id": 0,
27.                    "identifier": "feature",
28.                    "name": "input featurecoverage",
29.                    "type": "geom",
30.                    "description": "input featurecoverage with any domain",
31.                    "optional": false,
32.                    "url": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=Ge
tObservation&version=1.0.0&observedProperty=Rainfall_sensors&offering=rainfall_SEN
SORS",
33.                    "value": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=
GetObservation&version=1.0.0&observedProperty=Rainfall_sensors&offering=rainfall_SEN
SORS"
34.                },
35.                {
36.                    "id": 1,
37.                    "identifier": "attribute",
38.                    "name": "attributes",
39.                    "type": "string",
40.                    "description": "The attribute(s) of the featurecoverage whose values are interpolated",
41.                    "optional": false,
42.                    "url": "",
43.                    "value": "Jul01;Jul02;Jul03;Jul04;Jul05;Jul06;Jul07;Jul08;Jul09;Jul10;Jul11;Jul12;Jul13;Jul
14;Jul15;Jul16;Jul17;Jul18;Jul19;Jul20;Jul21;Jul22;Jul23;Jul24;Jul25;Jul26;Jul27;Jul28;Jul29;Jul30;J
ul31"

```

```

44.     },
45.     {
46.         "id": 2,
47.         "identifier": "weight_function",
48.         "name": "weight function",
49.         "type": "string",
50.         "description": "The method of weight function to be applied. Either Inverse Distance
method or Linear distance method",
51.         "optional": true,
52.         "url": "",
53.         "value": "invDist"
54.     },
55.     {
56.         "id": 3,
57.         "identifier": "weight_exponent",
58.         "name": "weight exponent",
59.         "type": "string",
60.         "description": "value for weight exponent n to be used in the specified weight functio
n (real value, usually a value close to 1.0).",
61.         "optional": false,
62.         "url": "",
63.         "value": "1"
64.     },
65.     {
66.         "id": 4,
67.         "identifier": "limiting_distance",
68.         "name": "limiting distance",
69.         "type": "double",
70.         "description": "value for the limiting distance: points that are farther away from an out
put pixel than the limiting distance obtain weight zero",
71.         "optional": false,
72.         "url": "",
73.         "value": "1"
74.     },
75.     {
76.         "id": 5,
77.         "identifier": "georef",
78.         "name": "georeference",
79.         "type": "georeference",
80.         "description": "the parameter can either be a georeference or the x extent of the the to
be created raster",
81.         "optional": false,
82.         "url": "",
83.         "value": "afrialiance.grf"
84.     }
85. ],
86. "outputs": [
87.     {
88.         "id": 0,
89.         "identifier": "result",
90.         "name": "result",
91.         "value": "",
92.         "description": "result",
93.         "type": "string"
94.     }

```

```

95.     ]
96.   },
97.   {
98.     "id": 1,
99.     "metadata": {
100.       "longname": "i3:Create Maplist",
101.       "label": "create_wps_maplist",
102.       "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
103.       "resource": "WPS",
104.       "description": "Create a maplist from a set of raster data.",
105.       "inputparametercount": 1,
106.       "outputparametercount": 1,
107.       "position": [
108.         394,
109.         278
110.       ]
111.     },
112.     "inputs": [
113.       {
114.         "id": 0,
115.         "identifier": "rasters",
116.         "name": "list of raster maps",
117.         "type": "string",
118.         "description": "A raster with multiple bands.",
119.         "optional": false,
120.         "url": "",
121.         "value": "0_to_0"
122.       }
123.     ],
124.     "outputs": [
125.       {
126.         "id": 0,
127.         "identifier": "raster",
128.         "name": "result",
129.         "value": "",
130.         "description": "result",
131.         "type": "maplist"
132.       }
133.     ]
134.   },
135.   {
136.     "id": 2,
137.     "metadata": {
138.       "longname": "i3:Table from GeoJSON",
139.       "label": "create_wps_table",
140.       "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
141.       "resource": "WPS",
142.       "description": "Create an ILWIS Table from a GeoJSON.",
143.       "inputparametercount": 2,
144.       "outputparametercount": 1,
145.       "position": [
146.         806,
147.         51
148.       ]
149.     },

```



```

150.     "inputs": [
151.         {
152.             "id": 0,
153.             "identifier": "feature",
154.             "name": "GeoJSON feature",
155.             "type": "geom",
156.             "description": "The url of the GeoJSON which is to be imported to ilwis tbt file",
157.             "optional": false,
158.             "url": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=Ge
tObservation&version=1.0.0&observedProperty=Rainfall_citizenpoints&offering=rainfall_CI
TIZENPOINTS",
159.             "value": "http://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=
GetObservation&version=1.0.0&observedProperty=Rainfall_citizenpoints&offering=rainfall_
CITIZENPOINTS"
160.         },
161.         {
162.             "id": 1,
163.             "identifier": "domain",
164.             "name": "Table Domain",
165.             "type": "string",
166.             "description": "The domain class to use",
167.             "optional": true,
168.             "url": "",
169.             "value": "wpdx.dom"
170.         }
171.     ],
172.     "outputs": [
173.         {
174.             "id": 0,
175.             "identifier": "result",
176.             "name": "ilwis table",
177.             "value": "",
178.             "description": "ilwis table",
179.             "type": "table"
180.         }
181.     ]
182. },
183. {
184.     "id": 3,
185.     "metadata": {
186.         "longname": "i3:PointMap From Table",
187.         "label": "pointmapfrom_wps_table",
188.         "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
189.         "resource": "WPS",
190.         "description": "Create an ILWIS Point map from table.",
191.         "inputparametercount": 4,
192.         "outputparametercount": 1,
193.         "position": [
194.             682,
195.             253
196.         ]
197.     },
198.     "inputs": [
199.         {
200.             "id": 0,

```

```

201.     "identifier": "table",
202.     "name": "Input table",
203.     "type": "table",
204.     "description": "The url of the input ilwis tbt file",
205.     "optional": false,
206.     "url": "",
207.     "value": "2_to_0"
208. },
209. {
210.     "id": 1,
211.     "identifier": "latitude_column",
212.     "name": "Latitude column",
213.     "type": "string",
214.     "description": "Column corresponding to latitude",
215.     "optional": false,
216.     "url": "",
217.     "value": "lat"
218. },
219. {
220.     "id": 2,
221.     "identifier": "longitude_column",
222.     "name": "Longitude column",
223.     "type": "string",
224.     "description": "Column corresponding to longitude",
225.     "optional": false,
226.     "url": "",
227.     "value": "lon"
228. },
229. {
230.     "id": 3,
231.     "identifier": "crs",
232.     "name": "coordinate system",
233.     "type": "string",
234.     "description": "Spatial Reference System e.g. LatlonWGS84",
235.     "optional": false,
236.     "url": "",
237.     "value": "wgs84"
238. }
239. ],
240. "outputs": [
241.     {
242.         "id": 0,
243.         "identifier": "result",
244.         "name": "Point Map",
245.         "value": "",
246.         "description": "Point Map",
247.         "type": "pointmap"
248.     }
249. ]
250. },
251. {
252.     "id": 4,
253.     "metadata": {
254.         "longname": "i3:Create Maplist2",
255.         "label": "create_wps_maplist2",

```

```

256.     "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
257.     "resource": "WPS",
258.     "description": "Create a maplist from a set of available rainfall maps. The rainfall map p
    roviders are CHIRPS, TAMSAT.",
259.     "inputparametercount": 3,
260.     "outputparametercount": 1,
261.     "position": [
262.         261,
263.         263
264.     ]
265. },
266. "inputs": [
267.     {
268.         "id": 0,
269.         "identifier": "startdate",
270.         "name": "Start date",
271.         "type": "date",
272.         "description": "The start date",
273.         "optional": false,
274.         "url": "",
275.         "value": "2015-06-30T22:00:00.000Z"
276.     },
277.     {
278.         "id": 1,
279.         "identifier": "enddate",
280.         "name": "End date",
281.         "type": "date",
282.         "description": "The end date",
283.         "optional": false,
284.         "url": "",
285.         "value": "2015-07-30T22:00:00.000Z"
286.     },
287.     {
288.         "id": 2,
289.         "identifier": "satelite",
290.         "name": "Satelite product e.g. CHIRPS",
291.         "type": "string",
292.         "description": "The rainfall satellite product, CHIRPS, TAMSAT",
293.         "optional": false,
294.         "url": "",
295.         "value": "chirps"
296.     }
297. ],
298. "outputs": [
299.     {
300.         "id": 0,
301.         "identifier": "raster",
302.         "name": "result",
303.         "value": "",
304.         "description": "result",
305.         "type": "maplist"
306.     }
307. ]
308. },
309. {

```

```

310.     "id": 5,
311.     "metadata": {
312.         "longname": "i3:Triple Collocation",
313.         "label": "triple_wps_collocation",
314.         "url": "http://130.89.221.193:82/WorkflowApp/app/api/wps.py?",
315.         "resource": "WPS",
316.         "description": "Triple Sensor Collocation can be used to validate 3 independent observ
ations at a location, when the error free true value is not known. With this you can judge, which
water or climate observation, i.e. your citizen observation, conventional station measurement o
r a remotely sensed satellite look-up and retrieval is most reliable.",
317.         "inputparametercount": 3,
318.         "outputparametercount": 1,
319.         "position": [
320.             521,
321.             445
322.         ]
323.     },
324.     "inputs": [
325.         {
326.             "id": 0,
327.             "identifier": "satelite_data",
328.             "name": "Satellite data",
329.             "type": "maplist",
330.             "description": "This is a map list of Earth observation data e.g. CHIRPS rainfall maps
",
331.             "optional": false,
332.             "url": "",
333.             "value": "4_to_0"
334.         },
335.         {
336.             "id": 1,
337.             "identifier": "station_data",
338.             "name": "Station data",
339.             "type": "maplist",
340.             "description": "This is a point map of In-situ or metereological station data",
341.             "optional": false,
342.             "url": "",
343.             "value": "1_to_1"
344.         },
345.         {
346.             "id": 2,
347.             "identifier": "citizen_data",
348.             "name": "Citizen data",
349.             "type": "pointmap",
350.             "description": "Point map of citizen generated data",
351.             "optional": false,
352.             "url": "",
353.             "value": "3_to_2"
354.         }
355.     ],
356.     "outputs": [
357.         {
358.             "id": 0,
359.             "identifier": "result",
360.             "name": "Evaluation report",

```

```

361.     "value": "",
362.     "description": "Evaluation report",
363.     "type": "geom"
364.   }
365. ]
366. }
367. ],
368. "connections": [
369.   {
370.     "fromOperationID": 0,
371.     "toOperationID": 1,
372.     "fromParameterID": 0,
373.     "toParameterID": 0
374.   },
375.   {
376.     "fromOperationID": 2,
377.     "toOperationID": 3,
378.     "fromParameterID": 0,
379.     "toParameterID": 0
380.   },
381.   {
382.     "fromOperationID": 4,
383.     "toOperationID": 5,
384.     "fromParameterID": 0,
385.     "toParameterID": 0
386.   },
387.   {
388.     "fromOperationID": 1,
389.     "toOperationID": 5,
390.     "fromParameterID": 0,
391.     "toParameterID": 1
392.   },
393.   {
394.     "fromOperationID": 3,
395.     "toOperationID": 5,
396.     "fromParameterID": 0,
397.     "toParameterID": 2
398.   }
399. ]
400. }
401. ]
402.}

```

Appendix C: BPMN Representation for Triple Sensor Water Accounting Workflow

```

1. <?xml version="1.0" encoding="UTF-8"?>
2. <bpmn2:definitions xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI" xmlns:camunda="http://camunda.org/schema/1.0/bpmn"
  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC" xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
  xmlns:ext="http://org.eclipse.bpmn2/ext" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-

```

```

instance" exporter="org.eclipse.bpmn2.modeler.core" exporterVersion="2018.2019_thesis" id
="Definitions_1" targetNamespace="http://org.eclipse.bpmn2/default/process">
3.   <bpmn2:itemDefinition id="ITEM_DEF_STRING" isCollection="false" structureRef="
xs:string" />
4.   <bpmn2:process id="_1" isExecutable="true" name="Subworkflow">
5.     <bpmn2:sequenceFlow id="SequenceFlow_Start" sourceRef="StartEvent_1" targetRef
="ServiceTask_2" />
6.     <bpmn2:sequenceFlow id="SequenceFlow_1" sourceRef="ServiceTask_0" targetRef="
ServiceTask_1" />
7.     <bpmn2:sequenceFlow id="SequenceFlow_2" sourceRef="ServiceTask_2" targetRef="
ServiceTask_3" />
8.     <bpmn2:sequenceFlow id="SequenceFlow_3" sourceRef="ServiceTask_4" targetRef="
ServiceTask_5" />
9.     <bpmn2:sequenceFlow id="SequenceFlow_4" sourceRef="ServiceTask_1" targetRef="
ServiceTask_5" />
10.    <bpmn2:sequenceFlow id="SequenceFlow_5" sourceRef="ServiceTask_3" targetRef="
ServiceTask_5" />
11.    <bpmn2:sequenceFlow id="SequenceFlow_End" sourceRef="ServiceTask_5" targetRef
="EndEvent_1" />
12.    <bpmn2:startEvent id="StartEvent_1" name="Start Workflow">
13.      <bpmn2:outgoing>SequenceFlow_Start</bpmn2:outgoing>
14.    </bpmn2:startEvent>
15.    <bpmn2:endEvent id="EndEvent_1" name="End Workflow">
16.      <bpmn2:incoming>SequenceFlow_End</bpmn2:incoming>
17.    </bpmn2:endEvent>
18.    <bpmn2:serviceTask id="ServiceTask_2" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:Table from GeoJSON" resource="WPS">
19.      <bpmn2:ioSpecification ioSpecification_="ioSpecification_2">
20.        <bpmn2:inputSet>
21.          <bpmn2:dataInputRefs>DataInput_GeoJSON feature_2</bpmn2:dataInputRe
fs>
22.          <bpmn2:dataInputRefs>DataInput_Table Domain_2</bpmn2:dataInputRefs>
23.        </bpmn2:inputSet>
24.        <bpmn2:dataInput id="DataInput_GeoJSON feature_2" itemSubjectRef="ITEM_
DEF_STRING" name="GeoJSON feature" optional="false" type="geom" value="http://13
0.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=GetObservation&version=1.
0.0&observedProperty=Rainfall_citizenpoints&offering=rainfall_CITIZENPOINTS" />
25.        <bpmn2:dataInput id="DataInput_Table Domain_2" itemSubjectRef="ITEM_DEF
_STRING" name="Table Domain" optional="true" type="string" value="wpdx.dom" />
26.      <bpmn2:outputSet>
27.        <bpmn2:dataOutputRefs>DataOutput_ilwis table_2</bpmn2:dataOutputRefs
>
28.      </bpmn2:outputSet>
29.      <bpmn2:dataOutput id="DataOutput_ilwis table_2" itemSubjectRef="ITEM_DEF_
STRING" name="ilwis table" type="table" value="" />
30.    </bpmn2:ioSpecification>
31.    <bpmn2:DataInputAssociation id="DataInputAssociation_GeoJSON feature_2">
32.      <bpmn2:sourceRef>GeoJSON feature</bpmn2:sourceRef>
33.      <bpmn2:targetRef>DataInput_GeoJSON feature_2</bpmn2:targetRef>
34.    </bpmn2:DataInputAssociation>
35.    <bpmn2:DataInputAssociation id="DataInputAssociation_Table Domain_2">
36.      <bpmn2:sourceRef>Table Domain</bpmn2:sourceRef>
37.      <bpmn2:targetRef>DataInput_Table Domain_2</bpmn2:targetRef>
38.    </bpmn2:DataInputAssociation>

```

```

39.     <bpmn2:DataOutputAssociation id="DataOutputAssociation_ilwis table_2">
40.         <bpmn2:sourceRef>ilwis table</bpmn2:sourceRef>
41.         <bpmn2:targetRef>DataOutput_ilwis table_2</bpmn2:targetRef>
42.     </bpmn2:DataOutputAssociation>
43.     <bpmn2:outgoing>SequenceFlow_3</bpmn2:outgoing>
44.     <bpmn2:incoming>SequenceFlow_Start</bpmn2:incoming>
45. </bpmn2:serviceTask>
46. <bpmn2:serviceTask id="ServiceTask_3" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:PointMap From Table" resource="WPS">
47.     <bpmn2:ioSpecification ioSpecification_="ioSpecification_3">
48.         <bpmn2:inputSet>
49.             <bpmn2:dataInputRefs>DataInput_Input table_3</bpmn2:dataInputRefs>
50.             <bpmn2:dataInputRefs>DataInput_Latitude column_3</bpmn2:dataInputRefs
51. >
52.             <bpmn2:dataInputRefs>DataInput_Longitude column_3</bpmn2:dataInputRe
fs>
53.             <bpmn2:dataInputRefs>DataInput_coordinate system_3</bpmn2:dataInputRe
fs>
54.         </bpmn2:inputSet>
55.         <bpmn2:dataInput id="DataInput_Input table_3" itemSubjectRef="ITEM_DEF_S
TRING" name="Input table" optional="false" type="table" value="2_to_0" />
56.         <bpmn2:dataInput id="DataInput_Latitude column_3" itemSubjectRef="ITEM_D
EF_STRING" name="Latitude column" optional="false" type="string" value="lat" />
57.         <bpmn2:dataInput id="DataInput_Longitude column_3" itemSubjectRef="ITEM_
DEF_STRING" name="Longitude column" optional="false" type="string" value="lon" />
58.         <bpmn2:dataInput id="DataInput_coordinate system_3" itemSubjectRef="ITEM_
DEF_STRING" name="coordinate system" optional="false" type="string" value="wgs84" /
>
59.         <bpmn2:outputSet>
60.             <bpmn2:dataOutputRefs>DataOutput_Point Map_3</bpmn2:dataOutputRefs
>
61.         </bpmn2:outputSet>
62.         <bpmn2:dataOutput id="DataOutput_Point Map_3" itemSubjectRef="ITEM_DEF
_STRING" name="Point Map" type="pointmap" value="" />
63.     </bpmn2:ioSpecification>
64.     <bpmn2:DataInputAssociation id="DataInputAssociation_Input table_3">
65.         <bpmn2:sourceRef>Input table</bpmn2:sourceRef>
66.         <bpmn2:targetRef>DataInput_Input table_3</bpmn2:targetRef>
67.     </bpmn2:DataInputAssociation>
68.     <bpmn2:DataInputAssociation id="DataInputAssociation_Latitude column_3">
69.         <bpmn2:sourceRef>Latitude column</bpmn2:sourceRef>
70.         <bpmn2:targetRef>DataInput_Latitude column_3</bpmn2:targetRef>
71.     </bpmn2:DataInputAssociation>
72.     <bpmn2:DataInputAssociation id="DataInputAssociation_Longitude column_3">
73.         <bpmn2:sourceRef>Longitude column</bpmn2:sourceRef>
74.         <bpmn2:targetRef>DataInput_Longitude column_3</bpmn2:targetRef>
75.     </bpmn2:DataInputAssociation>
76.     <bpmn2:DataInputAssociation id="DataInputAssociation_coordinate system_3">
77.         <bpmn2:sourceRef>coordinate system</bpmn2:sourceRef>
78.         <bpmn2:targetRef>DataInput_coordinate system_3</bpmn2:targetRef>
79.     </bpmn2:DataInputAssociation>
80.     <bpmn2:DataOutputAssociation id="DataOutputAssociation_Point Map_3">
81.         <bpmn2:sourceRef>Point Map</bpmn2:sourceRef>
82.         <bpmn2:targetRef>DataOutput_Point Map_3</bpmn2:targetRef>
83.     </bpmn2:DataOutputAssociation>

```



```

83.     <bpmn2:incoming>SequenceFlow_3</bpmn2:incoming>
84.     <bpmn2:outgoing>SequenceFlow_5</bpmn2:outgoing>
85.     </bpmn2:serviceTask>
86.     <bpmn2:serviceTask id="ServiceTask_0" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:Moving Average" resource="WPS">
87.     <bpmn2:ioSpecification ioSpecification_="ioSpecification_0">
88.     <bpmn2:inputSet>
89.     <bpmn2:dataInputRefs>DataInput_input featurecoverage_0</bpmn2:dataInput
Refs>
90.     <bpmn2:dataInputRefs>DataInput_attributes_0</bpmn2:dataInputRefs>
91.     <bpmn2:dataInputRefs>DataInput_weight function_0</bpmn2:dataInputRefs
>
92.     <bpmn2:dataInputRefs>DataInput_weight exponent_0</bpmn2:dataInputRefs
>
93.     <bpmn2:dataInputRefs>DataInput_limiting distance_0</bpmn2:dataInputRefs
>
94.     <bpmn2:dataInputRefs>DataInput_georeference_0</bpmn2:dataInputRefs>
95.     </bpmn2:inputSet>
96.     <bpmn2:dataInput id="DataInput_input featurecoverage_0" itemSubjectRef="ITE
M_DEF_STRING" name="input featurecoverage" optional="false" type="geom" value="htt
p://130.89.8.26/WorkflowApp/app/api/sos.py?service=SOS&request=GetObservation&vers
ion=1.0.0&observedProperty=Rainfall_sensors&offering=rainfall_SENSORS" />
97.     <bpmn2:dataInput id="DataInput_attributes_0" itemSubjectRef="ITEM_DEF_ST
RING" name="attributes" optional="false" type="string" value="Jul01;Jul02;Jul03;Jul04;Jul05;
Jul06;Jul07;Jul08;Jul09;Jul10;Jul11;Jul12;Jul13;Jul14;Jul15;Jul16;Jul17;Jul18;Jul19;Jul20;Jul21;Jul2
2;Jul23;Jul24;Jul25;Jul26;Jul27;Jul28;Jul29;Jul30;Jul31" />
98.     <bpmn2:dataInput id="DataInput_weight function_0" itemSubjectRef="ITEM_DE
F_STRING" name="weight function" optional="true" type="string" value="invDist" />
99.     <bpmn2:dataInput id="DataInput_weight exponent_0" itemSubjectRef="ITEM_D
EF_STRING" name="weight exponent" optional="false" type="string" value="1" />
100.    <bpmn2:dataInput id="DataInput_limiting distance_0" itemSubjectRef="ITEM_D
EF_STRING" name="limiting distance" optional="false" type="double" value="1" />
101.    <bpmn2:dataInput id="DataInput_georeference_0" itemSubjectRef="ITEM_DEF_
STRING" name="georeference" optional="false" type="georeference" value="afrialiance.grf"
/>
102.    <bpmn2:outputSet>
103.    <bpmn2:dataOutputRefs>DataOutput_result_0</bpmn2:dataOutputRefs>
104.    </bpmn2:outputSet>
105.    <bpmn2:dataOutput id="DataOutput_result_0" itemSubjectRef="ITEM_DEF_STR
ING" name="result" type="string" value="" />
106.    </bpmn2:ioSpecification>
107.    <bpmn2:DataInputAssociation id="DataInputAssociation_input featurecoverage_0"
>
108.    <bpmn2:sourceRef>input featurecoverage</bpmn2:sourceRef>
109.    <bpmn2:targetRef>DataInput_input featurecoverage_0</bpmn2:targetRef>
110.    </bpmn2:DataInputAssociation>
111.    <bpmn2:DataInputAssociation id="DataInputAssociation_attributes_0">
112.    <bpmn2:sourceRef>attributes</bpmn2:sourceRef>
113.    <bpmn2:targetRef>DataInput_attributes_0</bpmn2:targetRef>
114.    </bpmn2:DataInputAssociation>
115.    <bpmn2:DataInputAssociation id="DataInputAssociation_weight function_0">
116.    <bpmn2:sourceRef>weight function</bpmn2:sourceRef>
117.    <bpmn2:targetRef>DataInput_weight function_0</bpmn2:targetRef>
118.    </bpmn2:DataInputAssociation>
119.    <bpmn2:DataInputAssociation id="DataInputAssociation_weight exponent_0">

```



```

120.     <bpmn2:sourceRef>weight exponent</bpmn2:sourceRef>
121.     <bpmn2:targetRef>DataInput_weight exponent_0</bpmn2:targetRef>
122. </bpmn2:DataInputAssociation>
123. <bpmn2:DataInputAssociation id="DataInputAssociation_limiting distance_0">
124.     <bpmn2:sourceRef>limiting distance</bpmn2:sourceRef>
125.     <bpmn2:targetRef>DataInput_limiting distance_0</bpmn2:targetRef>
126. </bpmn2:DataInputAssociation>
127. <bpmn2:DataInputAssociation id="DataInputAssociation_georeference_0">
128.     <bpmn2:sourceRef>georeference</bpmn2:sourceRef>
129.     <bpmn2:targetRef>DataInput_georeference_0</bpmn2:targetRef>
130. </bpmn2:DataInputAssociation>
131. <bpmn2:DataOutputAssociation id="DataOutputAssociation_result_0">
132.     <bpmn2:sourceRef>result</bpmn2:sourceRef>
133.     <bpmn2:targetRef>DataOutput_result_0</bpmn2:targetRef>
134. </bpmn2:DataOutputAssociation>
135. <bpmn2:outgoing>SequenceFlow_1</bpmn2:outgoing>
136. </bpmn2:serviceTask>
137. <bpmn2:serviceTask id="ServiceTask_1" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:Create Maplist" resource="WPS">
138.     <bpmn2:ioSpecification ioSpecification_="ioSpecification_1">
139.         <bpmn2:inputSet>
140.             <bpmn2:dataInputRefs>DataInput_list of raster maps_1</bpmn2:dataInputRef
s>
141.         </bpmn2:inputSet>
142.         <bpmn2:dataInput id="DataInput_list of raster maps_1" itemSubjectRef="ITEM_D
EF_STRING" name="list of raster maps" optional="false" type="string" value="0_to_0" />
143.         <bpmn2:outputSet>
144.             <bpmn2:dataOutputRefs>DataOutput_result_1</bpmn2:dataOutputRefs>
145.         </bpmn2:outputSet>
146.         <bpmn2:dataOutput id="DataOutput_result_1" itemSubjectRef="ITEM_DEF_STR
ING" name="result" type="maplist" value="" />
147.     </bpmn2:ioSpecification>
148.     <bpmn2:DataInputAssociation id="DataInputAssociation_list of raster maps_1">
149.         <bpmn2:sourceRef>list of raster maps</bpmn2:sourceRef>
150.         <bpmn2:targetRef>DataInput_list of raster maps_1</bpmn2:targetRef>
151.     </bpmn2:DataInputAssociation>
152.     <bpmn2:DataOutputAssociation id="DataOutputAssociation_result_1">
153.         <bpmn2:sourceRef>result</bpmn2:sourceRef>
154.         <bpmn2:targetRef>DataOutput_result_1</bpmn2:targetRef>
155.     </bpmn2:DataOutputAssociation>
156. <bpmn2:incoming>SequenceFlow_1</bpmn2:incoming>
157. <bpmn2:outgoing>SequenceFlow_5</bpmn2:outgoing>
158. </bpmn2:serviceTask>
159. <bpmn2:serviceTask id="ServiceTask_4" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:Create Maplist2" resource="WPS">
160.     <bpmn2:ioSpecification ioSpecification_="ioSpecification_4">
161.         <bpmn2:inputSet>
162.             <bpmn2:dataInputRefs>DataInput_Start date_4</bpmn2:dataInputRefs>
163.             <bpmn2:dataInputRefs>DataInput_End date_4</bpmn2:dataInputRefs>
164.             <bpmn2:dataInputRefs>DataInput_Satellite product e.g. CHIRPS_4</bpmn2:dat
aInputRefs>
165.         </bpmn2:inputSet>

```

```

166.     <bpmn2:dataInput id="DataInput_Start date_4" itemSubjectRef="ITEM_DEF_ST
RING" name="Start date" optional="false" type="date" value="2015-06-
30T22:00:00.000Z" />
167.     <bpmn2:dataInput id="DataInput_End date_4" itemSubjectRef="ITEM_DEF_STR
ING" name="End date" optional="false" type="date" value="2015-07-
30T22:00:00.000Z" />
168.     <bpmn2:dataInput id="DataInput_Satelite product e.g. CHIRPS_4" itemSubjectRef
="ITEM_DEF_STRING" name="Satellite product e.g. CHIRPS" optional="false" type="stri
ng" value="chirps" />
169.     <bpmn2:outputSet>
170.         <bpmn2:dataOutputRefs>DataOutput_result_4</bpmn2:dataOutputRefs>
171.     </bpmn2:outputSet>
172.     <bpmn2:dataOutput id="DataOutput_result_4" itemSubjectRef="ITEM_DEF_STR
ING" name="result" type="maplist" value="" />
173. </bpmn2:ioSpecification>
174. <bpmn2:DataInputAssociation id="DataInputAssociation_Start date_4">
175.     <bpmn2:sourceRef>Start date</bpmn2:sourceRef>
176.     <bpmn2:targetRef>DataInput_Start date_4</bpmn2:targetRef>
177. </bpmn2:DataInputAssociation>
178. <bpmn2:DataInputAssociation id="DataInputAssociation_End date_4">
179.     <bpmn2:sourceRef>End date</bpmn2:sourceRef>
180.     <bpmn2:targetRef>DataInput_End date_4</bpmn2:targetRef>
181. </bpmn2:DataInputAssociation>
182. <bpmn2:DataInputAssociation id="DataInputAssociation_Satelite product e.g. CHIR
PS_4">
183.     <bpmn2:sourceRef>Satellite product e.g. CHIRPS</bpmn2:sourceRef>
184.     <bpmn2:targetRef>DataInput_Satelite product e.g. CHIRPS_4</bpmn2:targetRef
>
185. </bpmn2:DataInputAssociation>
186. <bpmn2:DataOutputAssociation id="DataOutputAssociation_result_4">
187.     <bpmn2:sourceRef>result</bpmn2:sourceRef>
188.     <bpmn2:targetRef>DataOutput_result_4</bpmn2:targetRef>
189. </bpmn2:DataOutputAssociation>
190. <bpmn2:outgoing>SequenceFlow_5</bpmn2:outgoing>
191. </bpmn2:serviceTask>
192. <bpmn2:serviceTask id="ServiceTask_5" implementation="http://130.89.221.193:82/
WorkflowApp/app/api/wps.py?" name="i3:Triple Collocation" resource="WPS">
193.     <bpmn2:ioSpecification ioSpecification_="ioSpecification_5">
194.         <bpmn2:inputSet>
195.             <bpmn2:dataInputRefs>DataInput_Satellite data_5</bpmn2:dataInputRefs>
196.             <bpmn2:dataInputRefs>DataInput_Station data_5</bpmn2:dataInputRefs>
197.             <bpmn2:dataInputRefs>DataInput_Citizen data_5</bpmn2:dataInputRefs>
198.         </bpmn2:inputSet>
199.         <bpmn2:dataInput id="DataInput_Satellite data_5" itemSubjectRef="ITEM_DEF_
STRING" name="Satellite data" optional="false" type="maplist" value="4_to_0" />
200.         <bpmn2:dataInput id="DataInput_Station data_5" itemSubjectRef="ITEM_DEF_S
TRING" name="Station data" optional="false" type="maplist" value="1_to_1" />
201.         <bpmn2:dataInput id="DataInput_Citizen data_5" itemSubjectRef="ITEM_DEF_S
TRING" name="Citizen data" optional="false" type="pointmap" value="3_to_2" />
202.         <bpmn2:outputSet>
203.             <bpmn2:dataOutputRefs>DataOutput_Evaluation report_5</bpmn2:dataOutp
utRefs>
204.         </bpmn2:outputSet>
205.         <bpmn2:dataOutput id="DataOutput_Evaluation report_5" itemSubjectRef="ITE
M_DEF_STRING" name="Evaluation report" type="geom" value="" />

```

```

206.    </bpmn2:ioSpecification>
207.    <bpmn2:DataInputAssociation id="DataInputAssociation_Satellite data_5">
208.      <bpmn2:sourceRef>Satellite data</bpmn2:sourceRef>
209.      <bpmn2:targetRef>DataInput_Satellite data_5</bpmn2:targetRef>
210.    </bpmn2:DataInputAssociation>
211.    <bpmn2:DataInputAssociation id="DataInputAssociation_Station data_5">
212.      <bpmn2:sourceRef>Station data</bpmn2:sourceRef>
213.      <bpmn2:targetRef>DataInput_Station data_5</bpmn2:targetRef>
214.    </bpmn2:DataInputAssociation>
215.    <bpmn2:DataInputAssociation id="DataInputAssociation_Citizen data_5">
216.      <bpmn2:sourceRef>Citizen data</bpmn2:sourceRef>
217.      <bpmn2:targetRef>DataInput_Citizen data_5</bpmn2:targetRef>
218.    </bpmn2:DataInputAssociation>
219.    <bpmn2:DataOutputAssociation id="DataOutputAssociation_Evaluation report_5">
220.      <bpmn2:sourceRef>Evaluation report</bpmn2:sourceRef>
221.      <bpmn2:targetRef>DataOutput_Evaluation report_5</bpmn2:targetRef>
222.    </bpmn2:DataOutputAssociation>
223.    <bpmn2:incoming>SequenceFlow_5</bpmn2:incoming>
224.    <bpmn2:incoming>SequenceFlow_5</bpmn2:incoming>
225.    <bpmn2:incoming>SequenceFlow_5</bpmn2:incoming>
226.    <bpmn2:outgoing>SequenceFlow_End</bpmn2:outgoing>
227.  </bpmn2:serviceTask>
228. </bpmn2:process>
229. <bpmndi:BPMNDiagram id="BPMNDiagram_1">
230.   <bpmndi:BPMNPlane bpmnElement="Subworkflow" id="BPMNPlane_ServiceTask_1
231.   ">
232.     <bpmndi:BPMNShape bpmnElement="StartEvent_1" id="BPMNShape_StartEvent_
233.     1">
234.       <dc:Bounds height="36.0" width="36.0" x="5.0" y="62" />
235.     </bpmndi:BPMNShape>
236.     <bpmndi:BPMNShape bpmnElement="EndEvent_1" id="BPMNShape_EndEvent_
237.     1">
238.       <dc:Bounds height="36.0" width="36.0" x="557" y="555" />
239.     </bpmndi:BPMNShape>
240.     <bpmndi:BPMNShape bpmnElement="ServiceTask_2" id="BPMNShape_ServiceTas
241.     k_1">
242.       <dc:Bounds height="50.0" width="110.0" x="806" y="51" />
243.     </bpmndi:BPMNShape>
244.     <bpmndi:BPMNShape bpmnElement="ServiceTask_3" id="BPMNShape_ServiceTas
245.     k_2">
246.       <dc:Bounds height="50.0" width="110.0" x="682" y="253" />
247.     </bpmndi:BPMNShape>
248.     <bpmndi:BPMNShape bpmnElement="ServiceTask_0" id="BPMNShape_ServiceTas
249.     k_3">
250.       <dc:Bounds height="50.0" width="110.0" x="509" y="30" />
251.     </bpmndi:BPMNShape>
252.     <bpmndi:BPMNShape bpmnElement="ServiceTask_1" id="BPMNShape_ServiceTas
253.     k_4">
254.       <dc:Bounds height="50.0" width="110.0" x="394" y="278" />
255.     </bpmndi:BPMNShape>
256.     <bpmndi:BPMNShape bpmnElement="ServiceTask_4" id="BPMNShape_ServiceTas
257.     k_5">
258.       <dc:Bounds height="50.0" width="110.0" x="261" y="263" />
259.     </bpmndi:BPMNShape>

```

```

252.    <bpmndi:BPMNShape bpmnElement="ServiceTask_5" id="BPMNShape_ServiceTas
k_6">
253.        <dc:Bounds height="50.0" width="110.0" x="521" y="445" />
254.    </bpmndi:BPMNShape>
255.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_Start" id="BPMNEdge_Seque
nceFlow_1">
256.        <di:waypoint x="41.0" y="36.0" />
257.        <di:waypoint x="806" y="76" />
258.    </bpmndi:BPMNEdge>
259.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_End" id="BPMNEdge_Seque
nceFlow_2">
260.        <di:waypoint x="569" y="495" />
261.        <di:waypoint x="569" y="555" />
262.    </bpmndi:BPMNEdge>
263.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_1" id="BPMNEdge_Sequence
Flow_3">
264.        <di:waypoint x="557" y="80" />
265.        <di:waypoint x="442" y="278" />
266.    </bpmndi:BPMNEdge>
267.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_2" id="BPMNEdge_Sequence
Flow_4">
268.        <di:waypoint x="854" y="101" />
269.        <di:waypoint x="730" y="253" />
270.    </bpmndi:BPMNEdge>
271.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_3" id="BPMNEdge_Sequence
Flow_5">
272.        <di:waypoint x="309" y="313" />
273.        <di:waypoint x="569" y="445" />
274.    </bpmndi:BPMNEdge>
275.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_4" id="BPMNEdge_Sequence
Flow_6">
276.        <di:waypoint x="442" y="328" />
277.        <di:waypoint x="569" y="445" />
278.    </bpmndi:BPMNEdge>
279.    <bpmndi:BPMNEdge bpmnElement="SequenceFlow_5" id="BPMNEdge_Sequence
Flow_7">
280.        <di:waypoint x="730" y="303" />
281.        <di:waypoint x="569" y="445" />
282.    </bpmndi:BPMNEdge>
283. </bpmndi:BPMNPlane>
284. </bpmndi:BPMNDiagram>
285. </bpmn2:definitions>

```

Appendix D: Sample operations of selected GIS tools

```

1.  {
2.    "QGIS": [
3.      {
4.        "name": "Raster difference",
5.        "label": "saga:griddifference",
6.        "inputs": [
7.          "coverage",
8.          "coverage",
9.          "numeric"

```

```

10. },
11. "description": "Raster difference",
12. "keywords": [
13.   "subtract",
14.   "difference",
15.   "minus"
16. ],
17. "outputs": [
18.   "coverage"
19. ]
20. },
21. {
22.   "name": "Raster division",
23.   "label": "saga:griddivision",
24.   "inputs": [
25.     "coverage",
26.     "coverage",
27.     "numeric"
28. ],
29. "description": "Raster division",
30. "keywords": [
31.   "divide",
32.   "division",
33.   "quotient"
34. ],
35. "outputs": [
36.   "coverage"
37. ]
38. },
39. {
40.   "name": "Raster product",
41.   "label": "saga:gridsproduct",
42.   "inputs": [
43.     "coverage",
44.     "coverage",
45.     "numeric"
46. ],
47. "description": "Raster product",
48. "keywords": [
49.   "product",
50.   "multiply",
51.   "times"
52. ],
53. "outputs": [
54.   "coverage"
55. ]
56. },
57. {
58.   "name": "Rasters sum",
59.   "label": "saga:gridssum",
60.   "inputs": [
61.     "coverage",
62.     "coverage",
63.     "numeric"
64. ],
65. "description": "Rasters sum",
66. "keywords": [
67.   "sum",
68.   "add",
69.   "combine"

```

```

70.   },
71.   "outputs": [
72.     "coverage"
73.   ]
74. },
75. {
76.   "name": "Raster Calculator",
77.   "label": "saga:rastercalculator",
78.   "inputs": ["coverage", "coverage", "text"],
79.   "description": "Rasters Calculator",
80.   "keywords": ["formular", "mapcalc", "expression"],
81.   "outputs": ["coverage"]
82. },
83. {
84.   "name": "Polygon centroids",
85.   "label": "qgis:polygoncentroids",
86.   "inputs": [
87.     "geom"
88.   ],
89.   "description": "Polygon centroids",
90.   "keywords": [
91.     "centroid",
92.     "center",
93.     "middle"
94.   ],
95.   "outputs": [
96.     "geom"
97.   ]
98. },
99. {
100.  "name": "Buffer vectors",
101.  "label": "gdalogr:buffervectors",
102.  "inputs": ["geom", "numeric"],
103.  "description": "Buffer vectors",
104.  "keywords": ["buffer", "buffering"],
105.  "outputs": ["geom"]
106. }
107. ],
108. "ILWIS": [
109.  {
110.    "name": "Map Calc",
111.    "label": "mapcalc2",
112.    "inputs": ["coverage", "coverage", "operator"],
113.    "description": "Perform a raster calculation on two rasters based on the applied operator",
114.    "keywords": [ "subtract", "difference", "minus" ],
115.    "outputs": [ "coverage" ]
116.  },
117.  {
118.    "name": "Buffer",
119.    "label": "buffer",
120.    "inputs": ["geom", "numeric", "numeric", "text"],
121.    "description": "Buffer vectors",
122.    "keywords": [ "buffer", "buffering" ],
123.    "outputs": [ "geom" ]
124.  }
125. ]
126.}

```

Appendix E: Code snippet for Transformation of PIW to QGIS Workflow

```

1. def piwToQgisWorkflow(workflow):
2.     workflowJSON = json.loads(workflow)
3.     qgisJSON = {}
4.     values = {}
5.     # inputs
6.     inputs = {}
7.     algos = {}
8.     for operation in workflowJSON['workflows'][0]['operations']:
9.         consoleName = ""
10.        first = WorkflowUtils.searchOperation("QGIS", operation['metadata']['longname'].lower())
11.        second = WorkflowUtils.searchOperation("QGIS", operation['metadata']['description'].lower())
12.        third = WorkflowUtils.searchOperation("QGIS", operation['metadata']['label'].lower())
13.        all = {first["hits"]: first, second["hits"]: second, third["hits"]: third}
14.        keys = list(all.keys())
15.        outputType = ""
16.        if first["hits"] >= second["hits"]:
17.            consoleName = first["operation"]["label"]
18.            outputType = first["operation"]["outputs"][0]
19.        else:
20.            consoleName = second["operation"]["label"]
21.            outputType = second["operation"]["outputs"][0]
22.        params = {}
23.        for input in operation['inputs']:
24.            if input['optional'] == False:
25.                if "_to_" not in input['value'] and (input['type'] == 'geom' or input['type'] == 'coverage'):
26.                    inputs[input['name'] + str(operation["id"])] = {
27.                        "values": {
28.                            "pos": {
29.                                "values": {
30.                                    "x": operation["metadata"]["position"][0],
31.                                    "y": operation["metadata"]["position"][1]
32.                                },
33.                                "class": "point"
34.                            },
35.                            "param": {
36.                                "values": {
37.                                    "isAdvanced": False,
38.                                    "name": input['name'] + str(operation["id"]),
39.                                    "default": "",
40.                                    "value": "",
41.                                    "exported": "",
42.                                    "hidden": False,
43.                                    "optional": input['optional'],
44.                                    "description": input['description']
45.                                },
46.                                "class": "processing.core.parameters.ParameterRaster" if input[
47.                                    'type'] == "coverage" else "processing.core.
parameters.ParameterVector"
48.                            }
49.                        },
50.                        "class": "processing.modeler.ModelerAlgorithm.ModelerParameter"
51.                    }
52.                if input['type'] == "coverage":
53.                    inputs[input['name'] + str(operation["id"])]["values"]["param"]["values"]
54.                    "showSublayersDialog" = True
55.                if "GRIDS" in params:

```



```

56.         grids = params["GRIDS"]
57.         grids.append({
58.             "values": {
59.                 "name": input['name'] + str(operation["id"])
60.             },
61.             "class": "processing.modeler.ModelerAlgorithm.ValueFromInput"
62.         })
63.         params["GRIDS"] = grids
64.     else:
65.         params["_RESAMPLING"] = 3
66.         params["GRIDS"] = [
67.             {
68.                 "values": {
69.                     "name": input['name'] + str(operation["id"])
70.                 },
71.                 "class": "processing.modeler.ModelerAlgorithm.ValueFromInput"
72.             }
73.         ]
74.     elif input['type'] == "geom":
75.         inputs[input['name'] + str(operation["id"])]["values"]["param"]["values"]["shapetype"] = [
76.             -1]
77.         params["INPUT_LAYER"] = {
78.             "values": {
79.                 "name": input['name'] + str(operation["id"])
80.             },
81.             "class": "processing.modeler.ModelerAlgorithm.ValueFromInput"
82.         }
83.     elif "_to_" not in input['value'] and (input['type'] != 'geom' and input['type'] != 'coverage'):
84.         # if input["identifier"] == "":
85.         params[input["identifier"].upper()] = input["value"]
86.     if "_to_" in input['value']:
87.         fromOperID = input["value"].split("_to_")[0]
88.         fromOper = WorkflowUtils.getOperationByID(fromOperID,
89.             workflowJSON['workflows'][0]['operations'])
90.         # If output is coverage
91.         if outputType == 'coverage':
92.             if "GRIDS" in params:
93.                 grids = params["GRIDS"]
94.                 grids.append({
95.                     "values": {
96.                         "alg": fromOper['metadata']['longname'] + str(fromOper["id"]),
97.                         "output": fromOper['outputs'][0]['name'] + str(fromOper["id"])
98.                     },
99.                     "class": "processing.modeler.ModelerAlgorithm.ValueFromOutput"
100.                 })
101.                 params["GRIDS"] = grids
102.             else:
103.                 params["_RESAMPLING"] = 3
104.                 params["GRIDS"] = [
105.                     {
106.                         "values": {
107.                             "alg": fromOper['metadata']['longname'] + str(fromOper["id"]),
108.                             "output": fromOper['outputs'][0]['name'] + str(fromOper["id"])
109.                         },
110.                         "class": "processing.modeler.ModelerAlgorithm.ValueFromOutput"
111.                     }
112.                 ]
113.         elif outputType == 'geom':
114.             params["INPUT_LAYER"] = {

```



```

115.         "values": {
116.             "alg": fromOper['metadata']['longname'] + str(fromOper["id"]),
117.             "output": fromOper['outputs'][0]['name'] + str(fromOper["id"])
118.         },
119.         "class": "processing.modeler.ModelerAlgorithm.ValueFromOutput"
120.     }
121.     else: # Just because there is no output datatype, assume it is a geom field
122.         params["INPUT_LAYER"] = {
123.             "values": {
124.                 "alg": fromOper['metadata']['longname'] + str(fromOper["id"]),
125.                 "output": fromOper['outputs'][0]['name'] + str(fromOper["id"])
126.             },
127.             "class": "processing.modeler.ModelerAlgorithm.ValueFromOutput"
128.         }
129.
130.     outputs = {}
131.     for output in operation['outputs']:
132.         if output['type'] == 'geom' or outputType == 'geom':
133.             outputs['OUTPUT_LAYER'] = {
134.                 "values": {
135.                     "description": output['description'],
136.                     "pos": {
137.                         "values": {
138.                             "x": operation["metadata"]["position"][0],
139.                             "y": operation["metadata"]["position"][1]
140.                         },
141.                         "class": "point"
142.                     }
143.                 },
144.                 "class": "processing.modeler.ModelerAlgorithm.ModelerOutput"
145.             }
146.         elif output['type'] == 'coverage' or outputType == 'coverage':
147.             outputs["RESULT"] = {
148.                 "values": {
149.                     "description": output['description'],
150.                     "pos": {
151.                         "values": {
152.                             "x": operation["metadata"]["position"][0],
153.                             "y": operation["metadata"]["position"][1]
154.                         },
155.                         "class": "point"
156.                     }
157.                 },
158.                 "class": "processing.modeler.ModelerAlgorithm.ModelerOutput"
159.             }
160.
161.     longname = operation['metadata']['longname'] + str(operation["id"])
162.     algos[longname] = {
163.         "values": {
164.             "name": longname,
165.             "paramsFolded": True,
166.             "outputs": outputs,
167.             "outputsFolded": True,
168.             "pos": {
169.                 "values": {
170.                     "x": operation["metadata"]["position"][0],
171.                     "y": operation["metadata"]["position"][1]
172.                 },
173.                 "class": "point"
174.             },

```

```

175.         "dependencies": [],
176.         "params": params,
177.         "active": True,
178.         "consoleName": consoleName,
179.         "description": operation['metadata']['description']
180.     },
181.     "class": "processing.modeler.ModelerAlgorithm.Algorithm"
182. }
183.
184. values["inputs"] = inputs
185. # Description or Help Information
186. values["helpContent"] = {}
187. # Dont seem to know
188. values["group"] = workflowJSON['workflows'][0]['metadata']['longname']
189. values["name"] = workflowJSON['workflows'][0]['metadata']['longname']
190. # Algorithm or processes
191. values["algs"] = algos
192. qgisJSON["values"] = values
193. qgisJSON["class"] = "processing.modeler.ModelerAlgorithm.ModelerAlgorithm"
194.
195. return json.dumps(qgisJSON)

```

Appendix F: Setting up the System

The following software must be installed to reproduce the system. This demonstration was carried out in a Windows environment.

A. Apache Server

Download and install Apache HTTP server to C:\Apache24. The following instruction will guide through the installation and configuration process.

The default port for Apache Web Server is 80. However, our demonstration used port 82.

- First, ensure that you have installed the latest C++ Redistributable Visual Studio 2015. Download it from the link below: https://aka.ms/vs/15/release/VC_redist.x64.exe
- Download Apache 64-bit from the link below. The version used for this set up was Apache 2.4. <https://www.apachelounge.com/download/>
- Extract the zipped folder and copy it to the root of C:\. This will be C:\Apache24 depending on the version of Apache you have downloaded.
- Add “C:\Apache24” and “C:\Apache24\bin” to your system path. To add a folder to the system path.
- The next step will be to register Apache as a service. Open the command prompt as administrator.
Run this command “httpd.exe -k install.”
- Go to start and search for Services. If you followed the steps successfully, Apache would be listed as one of the services running.
- The project files can now be copied to C:\Apache24\htdocs folder. The destination folder should be **C:\Apache24\htdocs\WorkflowApp**.

B. Python

- Go to the link and select python 3.5* 32 bit. <https://www.python.org/downloads/>
- Once downloaded, click to install using the default settings. Python will be installed to the folder: C:\Users\YOUR_USERNAME\AppData\Local\Programs\Python\Python-35-32. Later on, after finishing your installation, you will have to change the path of the python installation in all the python files under the folder **C:\Apache24\htdocs\WorkflowApp\app\api**.
- Copy that path and add to your system path variable.
- The next steps will involve installing required Python modules.
- To install the modules, open your command prompt as admin and use the following command to install each of the modules.
- python -m pip install “module name.”
- The following is the list of modules that you will install.

<i>Module</i>	<i>Description</i>	<i>Command</i>
<i>psycopg2</i>	Module for connecting to Postgres Database	python -m pip install psycopg2
<i>requests</i>	Module for handling HTTP requests	python -m pip install requests
<i>GDAL</i>	Module for processing spatial data	python -m pip install gdal
<i>xmldict</i>	Converting XML to python dictionary	python -m pip install xmldict
<i>Numpy</i>	Used with GDAL to manipulate geospatial data, mainly rasters	python -m pip install numpy
<i>FLASK</i>	Module for creating REST API	python -m pip install flask python -m pip install flask_cors python -m pip install flask_restful

The configuration of Apache and Python

- Edit Apache’s config file, C:\Apache24\conf\httpd.conf and add the following lines under the tag.

```
AddHandler cgi-script .cgi .py
Options Indexes FollowSymLinks ExecCGI
```

The lines should appear as follows.

```
<Directory “C:/Apache24/htdocs”>
```

```
.....
AddHandler cgi-script .cgi .py
Options Indexes FollowSymLinks ExecCGI
.....
```

```
</Directory>
```

- Now go to services and restart Apache service.

C. Installation of PostgreSQL and PostGIS

1. Download PostgreSQL 10 for win x86-64 from [this](#) link. This is for a 64bit operating system.
2. Start the installation of PostgreSQL. The installation will prompt for a location. Use the default location. C:\Program Files\PostgreSQL\10. Click next
3. The next prompt will be for the data directory. It will inherit the previous settings to assign the data directory. Click next.
4. Set the root password for the database.
5. Click Next to accept the default port of 5432.
6. Next, you will be prompted for a default locale. Click next to accept it.
7. Click next to start the installation.
8. Once the installation is completed, you will be asked if you want to allow Stack Builder to download and install tools. Check the checkbox to agree and click finish.
9. From the window that pops up, select "PostgreSQL 10 on port 5432". Click Next.
10. The next prompt will ask you to select the application you would like to install. Select "Spatial Extensions." Choose the PostGIS extension you would like to install. Click Next.
11. The next prompt will ask you to review your selection and choose a download directory. Use the default download directory given. Click Next. Wait for your download to complete. Click Next and Agree to accept the Licence Agreement.
12. The next prompt will ask you to check the component you would like to install and uncheck those that you wouldn't want to install. Do not change anything. Click Next.
13. The next prompt will request you to choose the install location. Do not change anything. Click next.
14. Next, you will be prompted to specify database connection settings. Enter the password that you had specified in 4 above and then click Next. Wait for the installation to complete.
15. For all the confirm dialog that appears, click "Yes."
16. Your installation is completed. Click close and Finish.

D. Apache Tomcat

Before installing Tomcat, you need to install JAVA.

1. Go to the link below to download Java. Select 64-bit and the latest version of Java.
2. <https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
3. Once downloaded, install using the default settings. The Java will be installed to **C:\Program Files\Java**.
4. Now, you need to set JAVA Home to the system environment variables. To do this, locate the path of your Java installation. For this documentation, Java was installed to **C:\Program Files\Java\jdk1.8.0_151**. Your installation will have a different Java version depending on your choice.
5. Open Command Prompt (make sure you Run as administrator, so you're able to add a system environment variable).
6. Set the value of the environment variable to your JDK installation path as follows:
setx -m JAVA_HOME "C:\Program Files\Java\jdk1.8.0_151".
7. Finally add the **C:\Program Files\Java\jdk1.8.0_151\bin** to your system path variable.

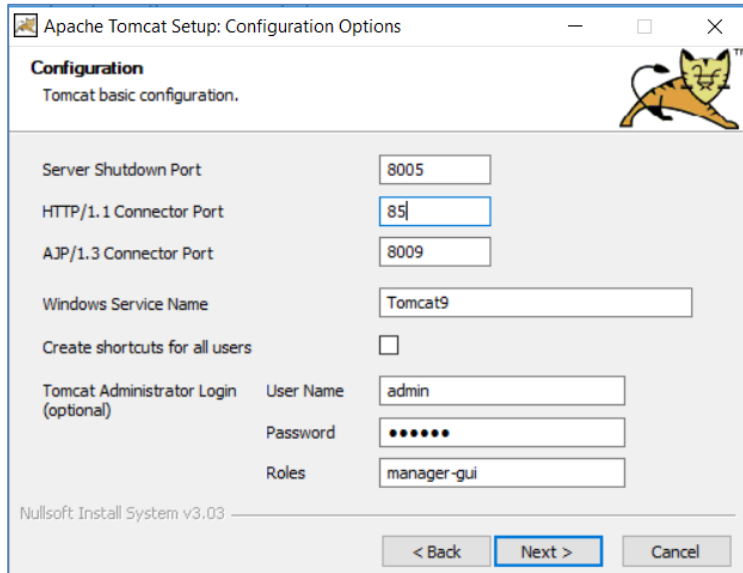
Now that Java has been installed and configured in your system, you can proceed and install Apache Tomcat.

- a) Download Apache Tomcat version 9 64-bit from the following link.
<http://www-us.apache.org/dist/tomcat/tomcat-9/v9.0.12/bin/apache-tomcat-9.0.12.exe>
- b) Click to install using the default settings. You will only need to change the following settings in the configuration options window.

HTTP/1.1 Connector Port to 85

User name: admin

Password: tomcat



- c) Set the path to your Java virtual machine (JVM). The default Java location will be selected. If no Java path is selected, you will have to add your Java location manually. Locate your Java JRE path. For my case, it is as follows:

C:\Program Files\Java\jdk1.8.0_151\jre

- d) Follow the default settings after that and finish the installation. Apache Tomcat will be added to your windows services automatically.
- e) The installation folder for Tomcat will be as follows.

C:\Program Files\Apache Software Foundation\Tomcat 9.0

Once we have installed Apache Tomcat successfully, the next is to install GeoServer.

E. GeoServer

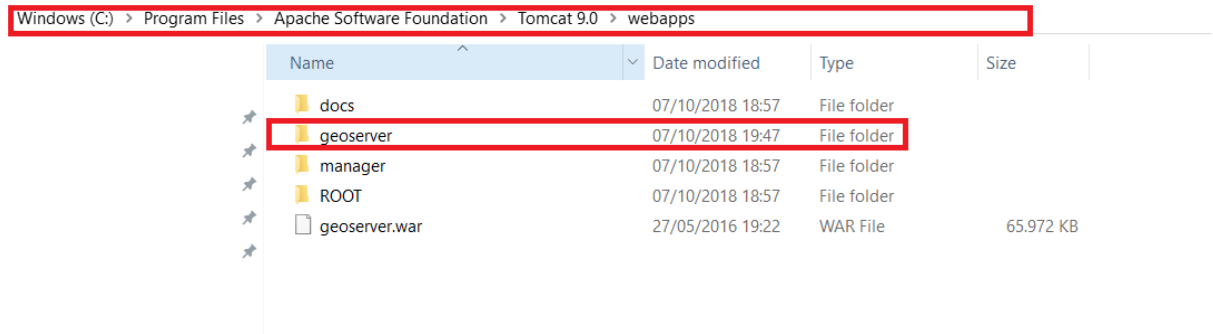
GeoServer is an open source server for sharing geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards. It implements several open standards which include Web Feature Services (WFS), Web Map Services (WMS) and Web Coverage Services (WCS) among others.

Installation of GeoServer

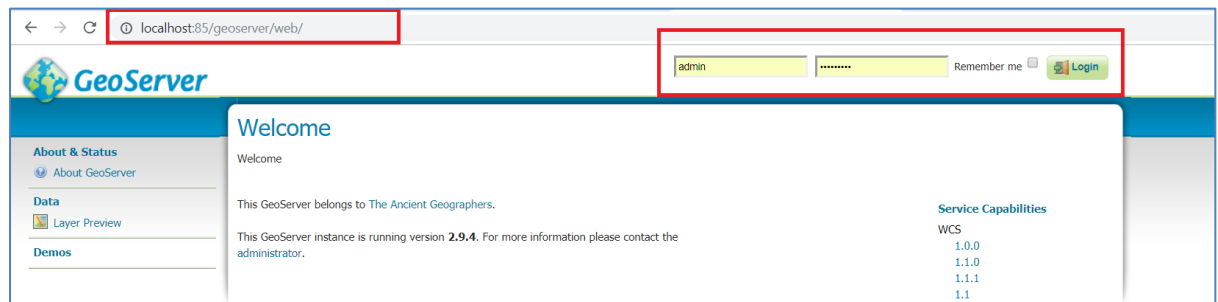
1. Download GeoServer from the following link. The version we will use is 2.9.4.
<http://sourceforge.net/projects/geoserver/files/GeoServer/2.9.4/geoserver-2.9.4-war.zip>
2. Extract the zipped file.
3. Inside the extracted folder, you will find **geoserver.war** file. Copy this file to the folder specified by the path below.

C:\Program Files\Apache Software Foundation\Tomcat 9.0\webapps

- The **geoserver.war** file will be extracted automatically by the running Apache Tomcat service. Once extracted, a folder for GeoServer will be created. The directory will look as shown below.



- In case **geoserver.war** is not extracted, you need to check if Apache Tomcat is running. The issue could be that it is not running. As such, you will have to start it.
- Once the **geoserver.war** is extracted, go to the following link in your browser. This will open the GeoServer webpage.
<http://localhost:85/geoserver>



- Login with the default settings for GeoServer

User: admin

Password: geoserver

- Once logged in, you can change your password since the default password can be used to attack your system and you may lose your GeoServer data.

Adding WPS extension to GeoServer

Refer to the following link to add a WPS extension to your GeoServer installation.

<https://docs.geoserver.org/stable/en/user/services/wps/install.html>

F. ILWIS

Two ILWIS versions were installed because instability was found in the way they were handling some operations through the command line.

Download ILWIS version 3.8 from <https://github.com/52North/Ilwis3Downloads/releases/tag/v3.8.5>.

Extract and save the contents to **C:\ilwis38**.

Download ILWIS 3.8.5.2 from <ftp://ftp.itc.nl/pub/52n/AfriAlliance/software/ILWIS3852.zip>. Extract and save the content to **C:\ILWIS3852**.

G. GeoServerJavaApp

This application was developed to publish raster files to a GeoServer. Please download the file from <https://gisedu.itc.utwente.nl/student/s1906240/GeoServerJavaApp.zip>.

Extract and save to C:\GeoServerJavaApp.

Appendix G: System Configuration

Open the *config.json* file in **C:\Apache24\htdocs\WorkflowApp**. Edit this file with the settings which have been used in the installations above. Let the IP address corresponds to the IP address of the installation computer. The rainfall data used in our demonstration can be downloaded from the URL below.

<ftp://ftp.itc.nl/pub/52n/AfriAlliance/sampleddata/ilwisout.zip>.

```

1. {
2.   "database": [
3.     {
4.       "host": "130.89.221.193",
5.       "port": 5434,
6.       "user": "",
7.       "password": "",
8.       "name": ""
9.     }
10.  ],
11.  "ilwis": ["C:\\ilwis38", "C:\\ILWIS3852"],
12.  "working_dir" : "C:\\Apache24\\htdocs\\WorkflowApp\\app\\api",
13.  "input_dir" : "D:\\ilwisout",
14.  "output_dir" : "C:\\Apache24\\htdocs\\WorkflowApp\\app\\api\\files\\triplecol"
15.  ,
16.  "output_url" : "http://130.89.221.193:82/WorkflowApp/app/api/files/triplecol",
17.  "geojar_path" : "C:\\GeoServerJavaApp\\PublishRaster.jar"
18. }
```

The last step in the configuration is to start two services which are very important for the success of the entire application. The ILWIS engine executes ILWIS functions whereas the REST service provides an API used in execution and transformation of workflows.

Go to folder **C:\Apache24\htdocs\WorkflowApp\app\api**. Using your Python IDE (PyCharm is recommended), run the files **ilwis_engine.py** and **rest.py**.