Design and Orchestration of Web Processing Services as Service Chains

NZABANDORA APHRODIS
February, 2016

SUPERVISORS:
Dr. J.M. Morales
Dr. Ir. R.L.G. Lemmens

Design and Orchestration of Web Processing Services as Service Chains

NZABANDORA APHRODIS

Enschede, The Netherlands, February, 2016

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: Geo-Informatics

SUPERVISORS:
Dr. J.M. Morales
Dr. Ir. R.L.G. Lemmens

THESIS ASSESSMENT BOARD:
Chair: prof.dr. M.J. Kraak
External examiner: dr.ir. C.H.J. Lemmen; Kadaster
Supervisor: dr.J.M. Morales
Second supervisor: dr.ir. R.L.G. Lemmens

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

Currently, the standards and many technologies behind web services are being developed. It is now possible to use distributed, self-contained, and modular applications in day-to-day work activities. GIS applications are nowadays developing on a distributed architecture composed of independent and specialized geo-services, designed to offer distributed functionality over the web environment. However, it is necessary to move from monolithic, coarse-grained services to fully web applications capable of processing more than one task at a time. Such applications can be assembled from individual web services that are orchestrated according to user-defined requirements.

Moreover, an architectural design is needed to allow its combination for specifying one or more needs. In this research, we investigated on how BPMN and geoprocessing services (WPS) can work together to create workflow containing spatial service based organizational flowchart. We realize that for both standard to work together an application programming interface (API) is needed. This application is triggered when the workflow is being orchestrated and controls the communication between workflow and the remote services. This application also extracts the information from the tasks in the workflow and construct the appropriate request to instantiate the services. Therefore, An API was developed with required functions to identify BPMN tasks that represent geo-services activity in the workflow, within those functions API is able to read geo-service tasks properties in workflow and make appropriate service call through a developed python script. The order that the services are called is based on sequences of activities in the workflow.

The external python script defines a standard way of sending WPS execute request based on OGC standard. It takes parameters from API and sends WPS execute request to Geo-server which was used as WPS implementation server in this research. BPMN was used to describe the sequence flow of activities and their execution order at high-level. In addition, API describes the low-level description of services such service call based on sequence and execution order as it is in BPMN workflow and organizational flowchart. This method was tested with a proof-of-concept where an organizational workflow of the municipality was modelled with BPMN and the sequence of geo-tasks in workflow were chained based on our proposed approach. In the end, the user is able to visualize the result combining the involved services on the web page.

Keywords: BPMN, Web service, service chain, orchestration, web processing services (WPS), API, and process engine.

ACKNOWLEDGEMENTS

This is a great opportunity to express my deep and sincere gratitude to my first supervisor Dr. J.M. Morales for his supervision, unlimited time for constructive discussion and guidance throughout this research period. I would also like to thank my second supervisor Dr. Ir. R.L.G. Lemmens for sharing his idea with me, his support and suggestions during my research work

I would like to thank the government of Rwanda throughout the ministry of natural resources and Rwanda Natural resources authority (RNRA)/Land and Mapping department (LMD) for their financial support offered to me to pursue my Master of Science in Geo-Information Science and Earth Observation in the Netherlands.

Special thanks to Dr. Emmanuel Nkurunziza Director General of RNRA and Eng. Sagashya former Deputy Director General for LMD for their encouragement during my study

Special thanks to my GFM classmates and all friends for their friendship and support throughout this study. Many thanks to my family for their encouragement and moral support. To my mommy brothers and sister.

Thanks also to all Rwandan community in ITC, my friend Jean Paul Ngarambe, Olivier Nzamuye for their encouragement and moral support during my time stay in Netherlands.

Special Thanks to my Girlfriend Justine Mukarubuga for your love and encouragement without considering the distance between us.

Above all, many thanks to my God for everything

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API: Application Programming Interface
BPEL: Business Process Execution Language
BPM: Business Process Management
BPMN: Business Process Management and Notation
BPMS: Business Process Management Software
CGI: Common Graphical Interface
CSS: Cascading Style Sheet
GI: Geographic Information
GIS: Geographic Information System
GML: Geographic Markup Language
HTML: Hypertext Transfer Protocol
HTTP: Hyper-Text Transfer Protocol
ISO: International Organization for Standardization
KVP: Key-Value-Pair
OGC: Open Geospatial Consortium
OWS: OGC Web Service
REST: Representational State Transfer
SDI: Spatial Data Infrastructure
SOA: Service-Oriented Architecture
SOAP: Simple Object Access Protocol
UDDI: Universal Description Discovery and Integration
URL: Uniform Resource Locator
W3C: World Wide Web Consortium
WCS: Web Coverage Service
WFS: Web Feature Services
WFS-T: Transaction Web Feature Service
WMS: Web Map Service
WMTS: Web Map Tile Service
WPS: Web Processing Services
WS: Web Service
WSDL: Web Service Description Language
WWW: World Wide Web
XML: Extensible Markup Language
XPDL: XML Process Definition Language

1. INTRODUCTION

1.1 Motivation and Problem Statement

Currently, the standards and many technologies behind web services are being developed. It is now possible to use distributed, self-contained, and modular applications in day-to-day work activities. Geographic information applications are nowadays developing on a distributed architecture composed of independent and specialized geo-services, designed to offer distributed functionality over the web environment. However, it is necessary to move from monolithic, coarse-grained services to fully web applications capable of processing more than one task at a time. Such applications can be assembled from individual web services that are orchestrated according to user-defined requirements.

In order to deal with spatial questions, the geospatial information system has been used to extract spatial information from the geospatial database(s). However, every system has its own proprietary format, GIS does not seem like open and interoperable. Due to the advancement of the computing systems and innovation in technologies such as Service Oriented Architecture (SOA), GIS has been changed from stand-alone systems to a service based model, and its functionality can be consumed as service over the web.

The key aspect of SOA is service orchestration, Service orchestration is the process of coordination and arrangement of multiple services exposed as a single aggregate service for creating new application requirements (Stollberg & Zipf, 2007). Therefore, when a single service cannot fulfil the requirement, the combination of several services should be used to do the tasks. In geoprocessing solving the complex problem requires chaining more than one service, whereby the output of one service is an input of another service. However, the orchestration of services from different geospatial nodes requires dealing with heterogeneity issues that are mainly related to syntactic and semantic interoperability.
The fast development of internet technology and computing systems has increased the need for spatial information sharing and interoperability among heterogeneous spatial information systems over the web environment. Therefore, the use of spatial data infrastructure (SDI) facilitates Geo-information user community to access distributed data over the web environment(de By, Lemmens, & Morales, 2009).
Furthermore, in distributed geoprocessing web services play a key role in publishing and discovering geo-information resources such as geospatial data and processing tools over the web (Mukherjee & Ghosh, 2010). For this purpose, Open Geospatial Consortium (OGC) launched the Web Processing Service (WPS) standard (OGC, Mueller, & Pross, 2015). This standard defines a standardized interface that facilitates publishing, discovery and binding the geospatial processes. It also allows execution of geospatial processes over the web using image data formats or data exchange standards such as XML/GML. Moreover, WPS provides client access to pre-programmed computation models that operate on spatially referenced data.

A geospatial process means computation, an algorithm or a model that is made available as a service instance operating on spatially referenced raster or vector data (Meng, Xie, & Bian, 2010).
For that reason, in order to provide the complex functionality of web services to the users, a method and methodology for designing and assembly of services are needed.
There are a number of possibilities to arrange web processes; this includes Business Process Execution Language (BPEL) engines, which dependent on WSDL (Web Service Description Language), and BPMN etc. Moreover, most of the web services are not described by WSDL. So, a software developer who uses BPEL to orchestrate services needs to create a WSDL file for each individual participant services. In addition to that, orchestrating services using BPEL approach is a problem because it is not capable of transmitting binary data which is saved in response to Web Map Service GetMap or Web Coverage Service GetCoverage Request (Stollberg & Zipf, 2007).

Therefore, the goal of this research is to develop a method which will help users to use BPMN notation to chain geo-processes based on organizational workflow. The proposed method insists on how BPMN and OGC web processing services can understand each other to create a workflow containing geoprocessing services.

1.2 Research identification

1.2.1 Research objectives

The main objective of this research project is to design an orchestration of web processing services as service chains.
In addition, BPMN (Business Process Modelling Notation) standard is used for providing a standardized and comprehensible graphical notation, to represent the geo-processes and their workflow at a high level.
In order to meet the main objective the following sub-objectives are addressed:

1. To determine the requirement for the creation of service chains.
2. To use standardized and readily understandable graphical notation, to represent the geo-processes and their workflow in the process model.
3. To analyse and develop a mechanism to transform and orchestrate service chain into an executable workflow (SDI application)

1.2.2 Research questions

1. What are the requirements to create a service chain? (sub-objective 1)
2. How to use standardized and readily understandable graphical notation like (BPMN), to represent the geo-processes and their workflow? (sub-objective 2)
3. What are the procedures of chaining geo-services with BPMN? (sub-objective 2)
4. How can web service orchestration improve the exploitation of OGC services? (sub- objective 3)
5. How to build and orchestrate the service chain into an executable workflow? (sub- objective 3)

1.2.3 Innovation aimed at

The main objective of this thesis was to design a method to orchestrate workflow containing web processing services (WPS) as service chain by using BPMN. As a proof-of-concept, a workflow for showing the part of the urban area that can be affected by the expansion of forest in new Landuse planning was implemented.
Based on organizational workflow all processes and their corresponding inputs and outputs was modelled in BPMN notation using ProcessMaker modelling tool to define the sequence of activities and their execution order in the workflow. The integrated execution of the services supporting task is done by developed API and python script.

1.2.4 Related Work

Lots of research has been conducted, and several approaches for orchestrating and chaining distributed services were proposed in order to facilitate access and visualization of spatial data.
However, the integration of web processing services in the service chain remains questionable due to the complexity of service to the user based on their requirements. ISO have been proposed three types of service chaining which deal with handling the complexity of services to the users.
Those types are: user-controlled, aggregate, and workflow managed. But also, the methodology for chaining geo-services are needed for integration of both syntactic and semantic service description (Rob Lemmens et al. 2007 and R. Lemmens et al. 2006). "The basis of orchestrating geoprocessing is to facilitate the implementation of an expeditious application system that can handle spatial-analytic function on the web" (Meng et al. 2010 and Kagoyire, 2009), BPEL as an orchestration language provide the ability to call services, process responses and handle process variables, control structure and errors Yu et al., (2012) and Jordan & Alves, (2007a).

For instance, Mukherjee, Mukherjee, & Ghosh, (2011) uses the service chaining approach along with fuzzy engine to avail more accuracy in decision making. (Stollberg & Zipf, 2007 and Meng et al. 2010) investigated on the possibility of using WPS orchestration to respond to the disaster management questions. Campagna, (2014) investigate on orchestrating spatial planning process using the high-level business process management. He used Bonita BPM as BPMS in orchestrating WPS services in Planning Systems. In(Campagna, 2014), Bonita BPM was used to offer the possibilities of reusing the predefined connectors for several systems and application. In addition, Bonita BPM allows the creation of new connector from scratch that allows execution and the control of workflows based on BPMN standard. Bonita Engine API's allows also the creation of a custom connector that can interact with external services such spatial web services for enabling them to be chained by the BPMS.

Campagna, Ivanov, & Massa, (2014) also investigate in the orchestration of services with Common Gateway Interface (CGI) written in Python 2.7 for executing several geoprocessing services that are available on the WPS server. Here, the orchestration and chaining services within BPMS has performed trough custom connectors scripts developed for Bonita BPM community relying on python.
This research will be interested in orchestrating services with BPMN standard as an essential block of BPM which was not natively developed for handling WPS services. A new method was developed and tested with a concrete use case.

1.3 Method adopted

This section describes the approach used to address our research problem to attain the research objectives. To accomplish this research work, the five core workflows of software development life-cycle as explained in described in Teams, (2004): Requirement definition, analysis, design, implementation and test.

1. Service orchestration requirement definition: During the requirement definition, we started by studying literature related geoprocessing workflows, the functionalities of web services or geo-services. In addition, Business Process management and notation (BPMN) was studied in deep as a methodological and technical approach for geoprocessing workflow design and implementations in this research. The current version of web processing services (WPS) will be studied as well.
2. Chaining requirement analysis: during this step the functional and non-functional requirement for orchestrating OGC service such WPS and WFS using BPMN standard were captured and analysed for designing a geoprocessing workflow method using high-level business process management tools.
3. Process Design: based on defined and analysed requirement the service chaining method using BPMN notation was designed.
4. Method implementation: during the implementation, the designed chaining method during design phase will be transformed into a generic method for chaining of services functionality.
5. Chaining method testing: During this step the implemented method will be tested by using the application scenario as proof- of- concept.

As a proof-of-concept, an organizational workflow model was designed and analysed using BPMN elements. BPMN service task was used as an artefact that represents processing-unit (geo-service) in the process model at a high-level.
These artefacts must be defined in the sequence that the services they represent must be called. Additional, API was developed to identify those geo-services in workflow and triggers the service call through a developed python script.

1.4 Structure of the Thesis

The thesis is structured into six chapters:
Chapter 1: Problem statement and motivation
This chapter provides the outline of the research; describe the research motivation, research problem, research objectives, research questions, related works, and the method adopted.
Chapter2: Web service architecture and web Service Orchestration
In this chapter, the basic information on SOA, web services, OGC services specifically WPS specification, Mode of communication and high-level business process modelling literature were reviewed for the preparation of next chapter.
Chapter 3: Analysing Requirement of Extending BPMN to chain geoprocessing services (WPS)
In this chapter, the mechanism of putting service together and move them from a high-level (BPMN) service orchestration description to an operational workflow was analysed as service chaining requirement based on general literature from chapter 2. The result of this analysis was used to design chaining method for WPS processing functionality with BPMN.
Chapter 4: Procedures of chaining WPS processes with BPMN
In this chapter, a method to model geo-processes with BPMN was designed. BPMN is used to model processes at high-level and an API were developed to describe low-level service implementation details such as service call by identifying BPMN tasks representing geo-service and make a call through a developed python script.
Chapter 5: Implementation of WPS Orchestration Method
This chapter describes the implementation of system prototype; the method proposed in chapter 4 was implemented within a chosen scenario as proof of concept.
Chapter 6: Discussion, Conclusion, and Recommendation
It contains the conclusion of the research report, outlining the achievement in the regards to innovation and recommendation for future research.

2. WEB SERVICE ARCHITECTURE AND WEB SERVICE ORCHESTRATION

2.1 Introduction

While carrying out, their daily activities most organizations need to share different information and resources over the internet. To become shareable, these resources need to be designed and provided as web service. Moreover, an architectural design is needed to allow the combination services for satisfying one or more needs. Service-oriented architecture (SOA) is one of architectural pattern in computer software design, where each application components provide service to other components via a communication protocol, typically over a network. For an organization to run their business based on web services it is necessary to interconnect their business and their systems.
The chapter starts with service oriented architecture in section 2.2, web services and Geoservices in section 2.3, OGC Web processing services (WPS) in section 2.4, Web service orchestration in section 2.5, Business Process Management Notation (BPMN) in section 2.6 and the last section 2.7 is a summary.

2.2 Service Oriented Architecture (SOA)

Service Oriented Architecture is a design paradigm for software systems. It is defined differently according to the context of use. In the context of this research, SOA is defined as software architectural concept that defines the use of services to support user's software requirements (Josuttis, 2007a). By Josuttis, (2007b) all definitions of SOA agree that SOA is "a paradigm for improving flexibility".

Service Oriented Architecture provides the way for developing new business services by reusing existing components of the program within the enterprise rather than rewriting the codes from scratch and developing new infrastructure. In the context of business process management (BPM), Web service orchestration is a process of coordination and arrangement of multiple services exposed as a single service for creating new aggregate service in line with real application requirements (Stollberg & Zipf, 2007). A web service can be defined as a program that offers its functionality to different systems through a defined interface over open protocols such as UDDI, SOAP, and HTTP. We distinguish three components in the concept of SOA; Service provider, service directory and service consumer (see figure 2.1). In addition to these components, it has also three operations; publish, bind, and find (Josuttis, 2007a) (see figure 1).

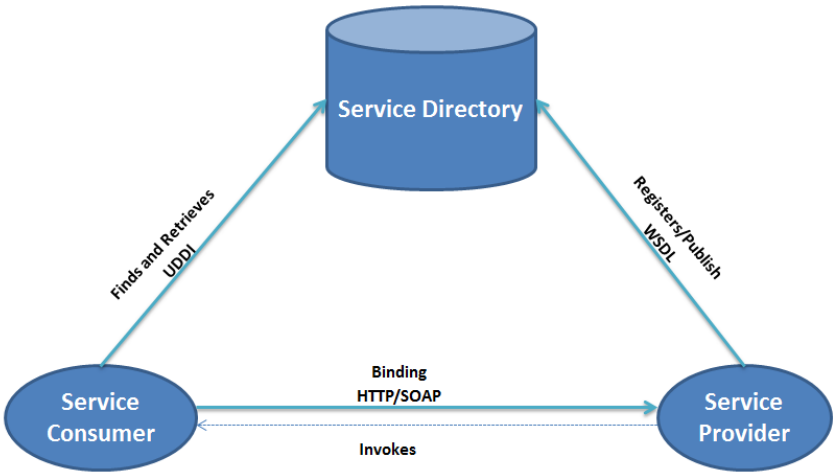


Figure 2.1: SOA Publish-Find-Bind paradigm

SOA relates the roles of these components with the operations to maintain automated discovery and the use of services.
The mentioned components can be described as the following:

- Service provider: publishes services to a registry and makes them available over the internet for the requests of customers.
- Service consumer: performs service discovery operations on the service registry in order to find the needed service, and accesses the services.
- Service directory helps service providers and service requesters to find each other by acting as a registry of the services.

The difference of SOA to classic modularization of program logic is that the functionality encapsulated by web service is delivered from business functions composing business processes(Booth et al., 2004).
For a business process, in order to operate in SOA environment services must have declaratively functional requirements and capabilities in agreed machine-readable format. According to Georgakopoulos, Ritter, & Benatallah, (2007), "the concept of SOA is not restricted to the technical side but also reaches out to the business process management (BPM)". That is because the functionality of web service is obtained from business functions that make up business processes, not from information technology systems.

SOA is concerned with the independent construction of services that can be combined into meaningful, high-level processes within the context of the Business (Georgakopoulos et al., 2007). SOA avoid the storage of the data used for a given processing activities in local copies and repositories of data. Furthermore, the algorithm used in one processing can be also used in others processing. Therefore, the use of SOA approach in the system can produce the systems flexible to the change of requirements, technologies, and offer easy system maintenance and more consistent of data and functionality. In this context, SOA is used as an abstract unit that allows sharing and encapsulating pieces of functionality and providing the way of exposing the processing functionality as a service over web environment.

2.3 Web Service and Geo-Service

2.3.1 Web Services

Different organizations and books provide a number of web services definitions. "Web services is defined as self-contained, modular, distributed, dynamic applications that can be distributed, published, and located" (Abdaldhem Albreshne, Patrik Fuhrer, 2009). Web services can be also invoked in a web environment to create processes and chains of operations within the process. In addition web services are a loosely-coupled function without paying attention to the platform implemented(Booth et al., 2004). Web services are built based on open standards such as HTTP, TCP/IP and XML (Abdaldhem Albreshne, Patrik Fuhrer, 2009). The concept of web service is basically needed for information and resources sharing to offer system reusability over the web environment. According to Josuttis, (2007) web service is the most preferred way of SOA realization. Its main goal is to achieve interoperability between different applications by using standards.

In addition, web service allows flexible integration of heterogeneous systems in a different domain such as B2C, B2B and organization applications integration using loosely coupled integration model as stated by Jordan & Alves, (2007a) and Abdaldhem Albreshne, Patrik Fuhrer, (2009).
In SOA, messaging protocol SOAP is used to enable interoperability between separately distributed systems. It basically standardizes the communication of distributed systems over the web. In general, SOAP is exchanged through HTTP which is a protocol used by internet browser to access web resources (Abdaldhem Albreshne, Patrik Fuhrer, 2009). For a service to be used it has to be discovered, UDDI enables businesses to list themselves on the internet. It generally enables business companies to interact and find each other in the web environment. The description of service is provided in Web Service Definition Language (WSDL). WSDL is W3C specification to ensure that all web services are described in consistent manner internet (see figure 2.2).

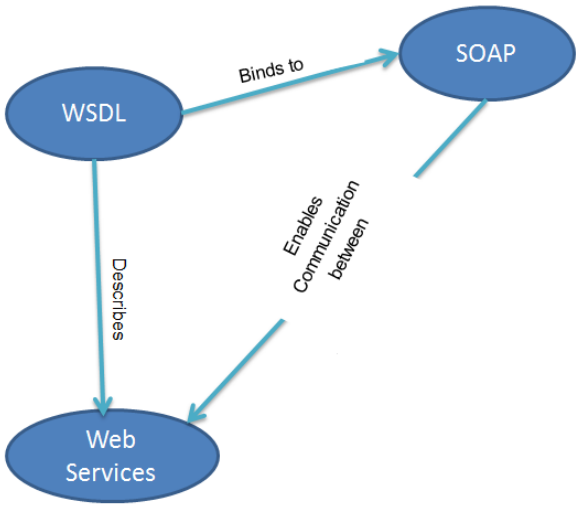


Figure 2.2: Web Service Communication (GIS HYDRO, 2009)

The potential integration of web services can be achieved when applications and business processes are able to integrate their complex interactions by using a standard process interaction model as explained in Jordan & Alves, (2007a) and Kudrass, (2003). This integration is also needed for the implementation of the complex model which involves several OGC services instances that can be chained to define a geoprocessing workflow for solving the complexity of real world problems. The example can be Disaster management and land use suitability analysis land use planning where different actors and several processing units involved in producing a suitable map for suitability analysis.

The Communication of Web Services

Web services perform functions; those functions can be from a simple request to complicated processes (Abdaldhem Albreshne, Patrik Fuhrer, 2009). After service deployment, it can be invoked and discovered by different applications or web services. Messaging protocol Simple Object Access Protocol (SOAP) is used to invoke web services. Simple Object Access Protocol messages are encoded in XML and transported over HTTP (Abdaldhem Albreshne, Patrik Fuhrer, 2009). The message exchange pattern is defined in a web service description language (WSDL). They are two communication modes in web services: Synchronous and Asynchronous.
In geospatial processing, some processes such as risk management and planning process require a combination of multiple services and deal with a large amount of data which takes a long execution time (Westerholt & Resch, 2015). Whereby, the communication mode is needed to inform the user different status of execution or where to take the result at the end of execution. The most of OGC standard and specifications are based on synchronous protocol even if the synchronous protocol is not enough to satisfy the more complex geoprocessing task (Hu, Yue, & Gong, 2013).

- ### Synchronous communication

In web services synchronous means every time that a client access web service application he receives a SOAP response. Synchronous communication mode in web services is request- response operation. In this type of communication, the client requires immediate response to the request (Westerholt & Resch, 2015). The OCG defines synchronous WPS as requirement class that indicate the general availability of synchronous capabilities on a WPS Server(OGC et al., 2015).
In synchronous communication, the connection of client remains open from the time the request is sent to the server; until the server sends back the response to the user (see figure 2.3). The advantage of this communication is that the client application knows the status of the requested service operation in very short time (Hu et al., 2013).

Moreover, it represents some limitations because the server deals with a large number of concurrent connections as each client maintains an open connection while waiting for the result (Westerholt & Resch, 2015).

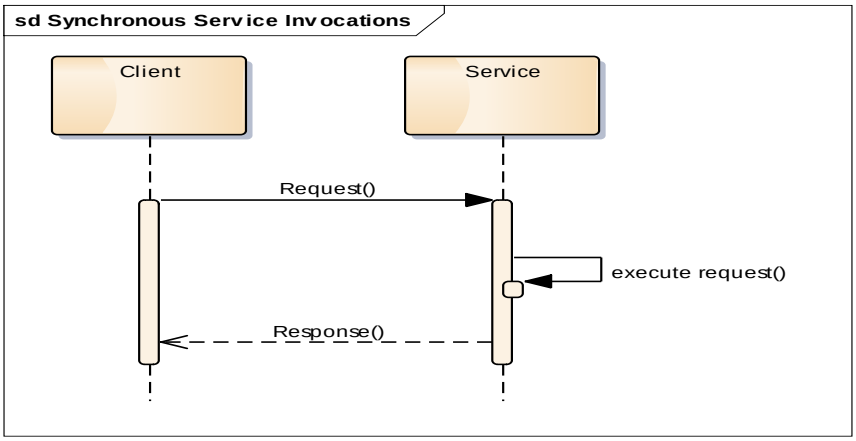


Figure 2.3: Synchronous service calls

- **Asynchronous communication**

In asynchronous, a user sent a request and he doesn't need to wait for the response. Whenever the response is ready he will receive a feedback call. Such communication behaviour is common to the services that require the complex processing that may take long processing time(Hu et al., 2013). In this communication, the situation might get worse when a sender sends a lot of asynchronous messages. Moreover, the order in which the user sends requests might be different from the order of receiving responses and the awaited response might not arrive at the time (Westerholt & Resch, 2015). Therefore, because sender and receiver are not synchronized the send need to wait until the response is available (see figure 2.4).

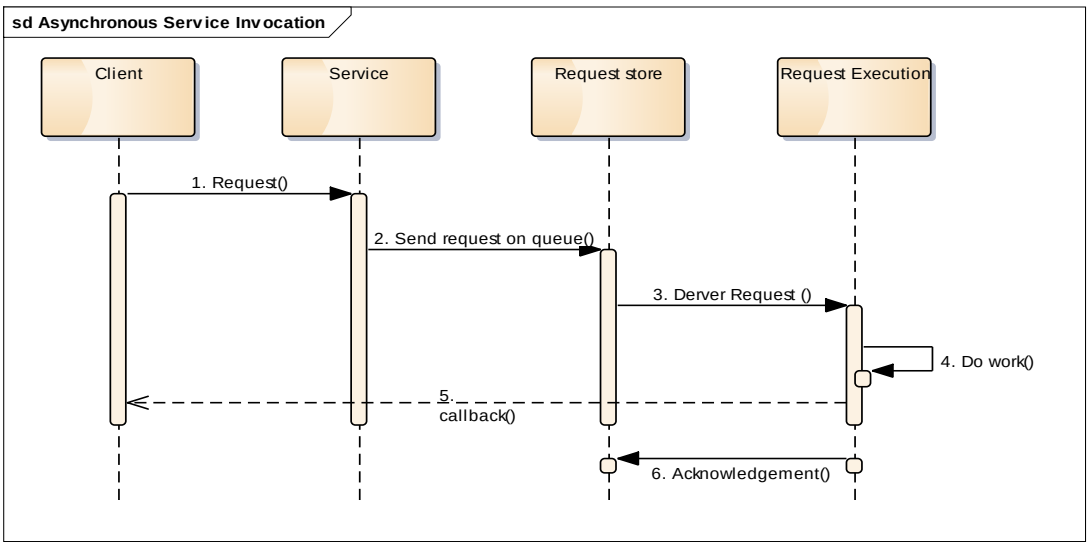


Figure 2.4: Sequence diagram of Asynchronous Service invocation

2.3.2 Geo-Services

Geoservices are normal web services that focus on geospatial information. Basically, geographic information comes from different sources and formats. Moreover, some processes such as spatial planning, disaster management require the use of geo-services to combine data different formats.
For that reason, SOA approach is applied to GIS domain where a number of standards have been launched. In fact, the evolution of technology moves GIS applications from standalone towards loosely coupled and distributed model based on specialized, self-contained, and interoperable geospatial web services. By Alameh, (2003), Geo-services can be grouped into different categories:

1. **Data Services**

Data access services are the services which provide access to actual spatial data sets. Data access service includes Web Coverage Services (WCS), Web Feature Services (WFS), and Web Feature Services-Transactional (WFS-T).

- **Web Coverage Services (WCS)**

This specification defines how WCS offers multi-dimensional coverage for access to the data over the internet(Baumann, 2012). By Baumann, this document also defines the basic requirements that WCS implementation must fulfil to support the retrieval of spatial data such as "coverage" to represent space/time-varying phenomena etc.
Different to WMS (Specification, 2001) and WFS which returns spatial data to be portrayed as a static map and spatial features respectively, WCS provides access to data coverage (Baumann, 2012).

- **Web Feature Services (WFS)**

This specification defines the way that geographic information is created, modified and exchanged on the internet through platform-independent calls(Date et al., 2015). In computing systems, OGC WFS provides an interface that allows the request for spatial features over the web(Date et al., 2015).
WFS is an OGC interface for discovery and providing access to geospatial feature data (vector data) in GML format (Meng et al., 2010). Further, data access provides access to Web Feature Services-Transactional (WFS-T) which enables editing feature geometry and related descriptive information.

2. **Portrayal Services**

- **Web Map Services (WMS)**

Web map service standard produces a map of georeferenced data from the geodatabase. It also serves image maps to one or more distributed spatial database by defining geographic layer and area of interest to be processed (Specification, 2001). This specification defines three operations: GetCapabilities which returns service metadata, GetMap which returns map image and finally, GetFeatureInfo which returns information on feature showed on the map.

- **OGC Web Map Tile Services (WMTS)**

Web map tile services provide a base solution to serve digital maps from predefined image tiles. This standard also implements the existing WMS, but WMS focuses on enabling users to obtain exactly the final image (OGC, 2010). WMTS advertises the available map tiles through its declaration in service metadata. This declaration defines the available tiles in each layer, style, format, scale, reference system and in all fragment of the covered area(OGC, 2010).

3. **Registry or catalogue service**

Registry or catalogue service provides a mechanism to register, search, describe, maintain and discovery to geospatial data and services to the users and applications (Senkler, 2007).

4. **Processing Services**

Open Geospatial Consortium (OGC) launched the Web Processing Service (WPS) standard. This standard defines a standardized interface that facilitates publishing, discovery and binding the geospatial processes(OGC et al., 2015). Moreover, WPS provides client access to pre-programmed computation models that operate on spatially referenced data (OGC et al., 2015).
In this research, we focused on WPS as our objective is to design an orchestration method for web processing services as service chains. Service chain in this context is an orchestration of web processing services where the output of the service executed first can cover at least one input of the next service(consider service as WPS operation or activity).

2.4 Web Processing Service (WPS)

As discussed in the previous section they are several number of OGC services but this research focuses on Web processing Services (WPS) as it is based on the orchestration of web processing services where different WPS instances need to be combined to answer user's questions. Web Features Services (WFS) was used to provide input features to WPS but is not more discussed. The main goal of WPS is to serve as host of one or more processing functionalities which can be accessed and executed in a web environment(OGC et al., 2015). Up to now, different WPS versions was published by OGC consortium but the current one is (OGC et al., 2015). The role of this standard is to define a standardized interface that provides the rules of how inputs and output (request and response) for geospatial processing services are manipulated. Examples of web processing services can be polygon overlay, coordinate transformation, buffering, intersection, and data conversion etc.

This interface plays a key role in spatial processing services because it provides all required information on how input and output of a specific operation are manipulated and treated as input to the next operation(OGC et al., 2015). By its methods, WPS defines how a client can request the execution of a process, and how the output from the process is manipulated. The standardized interface provided by OGC also facilitates publishing, discovery and binding the geospatial processes services which are also key for service orchestration see figure 2.1.

In data manipulation or processing, WPS plays a key role in allowing execution of geospatial processes over the web using different data exchange standards such as XML, GML etc. Moreover, in the context of information and resources sharing WPS provides client access to pre-programmed computation models that operate on the georeferenced dataset (Mukherjee & Ghosh, 2010). A geospatial process means a computation, an algorithm or a model that is made available as a service instance operating on spatially referenced raster or vector data (Meng et al., 2010).

2.4.1 Building WPS request

In the context of WPS implementation, it is necessary to know how to build a request for one or more operation for its description or execution. By (OGC et al., 2015), WPS is accessed from the web browser by the use of Hypertext Transfer Protocol (HTTP), which enables communication between client and server (see figure 2.3). This protocol uses two methods Get and Post for accessing WPS Server(OGC et al., 2015). HTTP Get is used for retrieving the targeted information from WPS server and HTTP Post is used to submit the data to be processed to specified resources. The current WPS specification defines six different operations: GetCapabilities, DescribeProcess, Execute, GetStatus, GetResult and Dismiss (OGC et al., 2015).

But in this research GetCapabilities, DescribeProcess and execute are only used for providing the basic information on available processes and metadata in the implementation of WPS orchestration in chapter five of this research.

WPS GetCapabilities operation

This operation is required to all OGC services. Its role in the context of WPS implementation is to provide access the basic service metadata such as service identification, service provider, operation metadata, and links to the operations and all processes offered by WPS implementation Server(OGC et al., 2015). This server has to support the GetCapabilities operation via HTTP GET or HTTP POST but the last one is optional (OGC et al., 2015). The response of this operation is an XML document called capabilities document. The Key–Value-Pair (KVP) of GetCapabilities request uses two mandatory parameters request and service. The KVP can be translated into an XML request submitted via HTTP POST to the WPS implementation Server. Moreover, both request (KVP or XML) return an XML document to inform the client about the processes being offered by WPS instance for more information refer to (OGC et al., 2015) (see table 2.1 for example).

Example of WPS GetCapabilities Request

http://Placeholder/geoserver/ows? service=WPS& version=2.0.0& request=Getcapabilities	Address of the Server Type of OGC web service Service version Request Type

Table 2.1: Example of GetCapabilities Request

WPS DescribeProcess operation

This operation is more important to WPS client who need to request full information on one or more processes that can be executed by the service in service orchestration or business process. This description includes the input (s), output (s) parameters and formats that can be used to automatically build a user interface and the parameter values to be used to execute needed processes (OGC et al., 2015).

Example of WPS DescribeProcess Request

By OGC Standard, this request inherits basic properties from the requestBaseType. It has an identifier to identify process with other processes (OGC et al., 2015). In case, the service supports multiple languages for describing the process the desired language of the free-text elements in the process description may be required for a language parameter. By(OGC et al., 2015) WPS server implements HTTP GET for transfer of DescribeProcess request, using KVP encoded (see table 2.2).

http://Placeholder /geo-server/ows? Request=DescribeProcess& Service=WPS& Version=2.0.0& Language=en& Identifier=intersection,union,buffer	Server Address Request Type Service Name Version of the Service Language Process to be requested

Table 2.2: An example of a DescribeProcess request via HTTP GET

This request also can be sent using HTTP POST to WPS Servers using XML encoding only (OGC et al., 2015) (see table 2.3).

```
<DescribeProcess service="WPS" version="1.0.0" language="en">
  <ows:Identifier>intersection</ows:Identifier>
  <ows:Identifier>union</ows:Identifier>
</DescribeProcess">
```

Table 2.3: An example of a DescribeProcess request via HTTP POST

WPS Execute Operation

WPS execute operation differ from WPS GetCapabilities and WPS DescribeProcess operations in the way that the request is sent to the server (OGC et al., 2015). It allows WPS users to run a specified process implemented by WPS implementation server. Therefore, the provided input parameter values return the output to WPS client after execution, Moreover, the inputs can be directly included in WPS execute the request or by reference (OGC et al., 2015).

Example of WPS Execute Request

WPS execute request provides support for multiple inputs, each input is referred to the forms of inputs as required for single WPS execute Request (OGC et al., 2015). The most used way of providing large inputs to WPS server is providing one or more URLs of inputs values for invoking WPS execute the request. Table 2.4 shows how WPS execute request can be sent to the WPS server using HTTP Get for transfer of the execute request by the use of KVP to encode the parameters(OGC et al., 2015).

http://Placeholder/geoserver/ows?	WPS Server Address
Request=Execute&	Request Name
Service=WPS&	Service Name
Version=2.0.0&	Version number
Language=en-CA&	Language
Identifier=Buffer&	Operation to be performed by WPS server
DataInputs=""&	
ResponseDocument=BufferedPoly&	KVP encoded inputs parameters
storeExecuteResponse=true&	Response document name
lineage=true&	Execute Store responde
status=true	

Table 2.4: An example of an Execute request via HTTP GET

In addition, WPS server uses a mandatory method HTTP POST. When this method is used to execute a process, an XML file containing inputs parameters is sent to the server. This method is also used when more than one inputs parameter are needed by the operation (OGC et al., 2015). In this research, we used HTTP POST method to send execute request to WPS Server.

2.4.2 WPS data types

The OGC standard defines three types of inputs and outputs: Literal data, ComplexData, and BoundingBoxData (OGC et al., 2015).

- **ComplexData**: This data type is used for pasting the complex data such as Vector. Raster and other data to the server like the URL of the GetFeature request. It does not describe the particular structure for value encoding.
  This type is a realization of the abstract DataDescription elements. Moreover, the complex data value is directly passed or returned by a process.

- **LiteralData** this type can be any character string of special type; it encodes atomic data such as String, Float, Integer and Boolean. It also inherits essential elements from OGC web service: DomainType. It should also use the well-known types from XML schema by their URIs.
- **BoundingBoxData** this type is used if someone needs to define or obtain some kind of bounding box like upper or lower coordinates.

2.5 Web Service Orchestration

In section 2.3 of this chapter, we explained how service is published, find and bind. Once a service is published it can be discovered by another web service or any application. It may happen that some questions require the combination of multiple services; in that case, service can be orchestrated and executed as service chain. The orchestration is the process of describing the automated arrangement, coordination and management of operations within a process. In the context of this research web service, the orchestration is the process of describing automated arrangement of services based on their execution order to solve user problem that requires the combination of multiple services. In addition, the orchestration is also defined as a process of interacting two or more application or web services together to automate a process.

In (Zhao, Di, & Yu, 2012), based on the standard of web service access and web service interaction, individual services can be reused and assembled into a service chain to represent high-level service workflow of activities. Workflow modelling is a technique that uses several description tools, mainly schemas and diagrams to describe processes inside the enterprise. Processes are series of activities that are executed in a predefined sequence ordered according to a set of process rules. According to Růžička, (2009) running process model means reading inputs, invoking web services for remote processing, decide based on the condition given by the modeller, repeating some part of the process where necessary for father operations.

In most case processes need to be transformed by intermediate languages such as Business Process Execution Language (BPEL) and BPMN. BPMN is a standard for graphically representing business processes but not geo-processes services. This language is rich semantically and allows representing activities (tasks/sub-processes), actors (pool, lanes) and a variety of execution constraints. Business workflow languages such as BPMN, BPEL and XPDL plays a complementary role to model at both high-level and low-level of the business workflow and geo-processes (Strickland, Whittington, Taylor, & Wang, 2006).

BPMN is essentially a graphical notation to build an abstract workflow with a set of high-level tasks to achieve the goal. Term "high-level" is used in this research to describe the overall goals and the features of the model. In addition, "low- level" is used to describe the individual components by providing all details. It is basically concerned with an individual component within the model.
BPEL and XPDL are aimed at describing the executable workflows, the specification of low-level implementation details such as calling a web services or geo-services in the process model, gathering their responses and handling the proper transformation operations as it is explained in Chinosi & Trombetta, (2012) and Zhao, Foerster, & Yue, (2012).

Campagna et al., (2014) defined how BPMN can be used to model geo-processes by developing a connector to support execution of distributed data such as Web feature Services (WFS), web processing services (WPS). By (Campagna, 2014), BPMN can be used in the different planning process for solving GI questions with iterative shifts from high-level general models to executable workflow. Web service orchestration works through message exchange in the domain layer. Since single service is not designed to communicate with other services of the different application. Therefore, messages must be exchanged between nodes according to predefined business logic and execution order so that the composite service systems or application can run as it is demanded by the end users.

A Web services enable this communication by using a combination of open protocols and standards like XML, SOAP and WSDL (Abdaldhem Albreshne, Patrik Fuhrer, 2009). It uses XML for tagging the data, Simple Object Access Protocol (SOAP) for sending and receiving messages and finally WSDL to describe the availability of the services (see figure 2.2). Service chaining can be done manually by feeding the output of one process to be the input of next process, semi-automatically by defining the sequence of web service interaction in a configuration file, or fully automatically by providing capabilities to establish a self-organizing-net. In geoprocessing, a process is an atomic service designed to perform one single WPS task. Therefore, in service orchestration and chaining, several processes are combined to carry out the activities. In this research, a new method of using BPMN to allow call and chaining geo-service such as WPS, WFS was developed and implemented with a proof-of-concept.

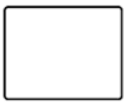
2.6 Business Process Management Notation (BPMN)

The BPMN specification is originally developed and conceived by Business Process Management Initiative (BPMI) but is currently maintained by Object Management Group (OMG) as graphical notation with clear semantics. This specification provides a graphical notation for expressing business process in a Business Process Diagram (BPD) (Object Management Group, Parida, & Mahapatra, 2011). This specification also presents the way of obtaining automatically XML code to deploy in workflow engines or to be shared. With this specification is possible to bind the graphical notation elements and constructs a block of structured process execution as explained in Object Management Group et al., (2011) and Allweyer, (2010).

BPMN is rich semantically and allows representing activities (tasks/sub-processes), actors (pool, lanes) and a variety of execution constraints. A task with the process can be automatic or mixed, and manual. The automated task can support the execution of distributed data such as Web feature Services (WFS), processing services like Web Processing services (WPS) as explained in Campagna et al., (2014).
In this research, BPMN is used to provide the basic elements to model WPS orchestration based on organizational workflow at a high level and to describe executable workflow or the processes at low-level implementation details through a developed API and python Script. BPMN provides several elements to build workflow in an abstract way and align the functional structure of tasks at high-level (see figure 2.5).

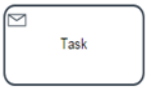
: The start event symbol signals the first step of a process

: The end event symbol signals the final step in a process



: A task is a unit of work – the job to be performed. It is an atomic activity within a process flow

Receive Task



: Waits for a Message to arrive from an external Participant Once the Message has been received, the Task is completed

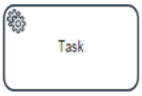
User Task

: A typical "workflow" Task where a human performer performs the Task with the assistance of a software application

Script Task

Is executed by a business process engine the modeler or implementer defines a script in a language that the engine can interpret. When the Task is ready to start, the engine will execute the script. When the script is completed, the Task will also be completed.

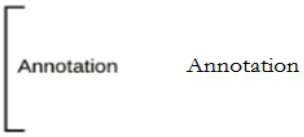
Parallel gateway



: A service task is any task that uses an automated application or web service to complete the task.

Sequence flow - connects flow objects in proper sequential order.

The data input symbol represents data requirements that tasks in the business process depend on.



Swim lanes

Figure 2.5: Common BPMN elements used throughout this research (Object Management Group et al., 2011)

Business process modelling has been an important topic in research communities (Foerster, Schaeffer, Brauner, & Jirka, 2009). Whereas, the use of web service technologies enabled different protocols to create process composition by integrating internal and external business processes including their interactions (Ying, Qunyong, & Linjun, 2012). The orchestration language like BPEL is the most common used technology to orchestrate web services (Donaubauer & Straub, 2010). Although, it has some limitation of interacting only with SOAP web services. Moreover, the graphical representation of BPEL does not provide a top level model for business but it provides a chain of web services. Therefore, the new BPMN specification answers this problem by defining a process in terms of business tasks with no limit to web services calls. Up today, BPMN standard is important for an organization to achieve business process interaction. Moreover, this specification has some limitation to be addressed may be in next version. This research focusses on how BPMN can be extended to allow the call of spatial services in the process model.

2.7 Summary

In this chapter, we studied literature related to different concepts need for implementation of our research basically functionalities of web services or geo-services in geo-information processing. In addition, Business Process Modelling and Notation (BPMN) were studied in deep as a methodological and technical approach for designing and implementing geoprocessing workflow. The current version of web processing services (WPS) was studied as well. The content of this chapter is necessary for two next chapters: analysing requirements of chaining WPS services and chaining WPS processes with BPMN as it provides the overview of how different technologies and theories are described in a scientific way.

3. ANALYSING REQUIREMENT OF CHAINING WEB PROCESSING SERVICES

3.1 Introduction

This chapter discusses and analyses the functional and non-functional requirements that are needed for chaining two or more web services or geo-services using business process management and notation (BPMN). We explored how BPMN notation can include geoprocessing services (WPS) in the workflows. In addition, we also explore different styles of web service orchestration. The chapter starts with web service orchestration style in section 3.2, moving from high-level service orchestration to low level service implementation details in section 3.3, procedures of service chaining WPS services with BPMN in section 3.4, and the last section 3.5 is a summary.

3.2 Web Service Orchestration Style

The evolution of business and technologies increases the number of systems that run businesses. In current processes, the real requirements for linking systems are needed. Today's technology of web services and web service orchestration offer a future-proof secured, accessible, and scalable mechanism for system communication and process reuse(Strickland et al., 2006). Moreover, these technologies are needed in spatial data processing when utilizing a large volume of data from one or multiple systems to carry out business process or to answer some geospatial questions that require the combination of multiple services.

The term "high-level orchestration" is used in this research to describe the overall goals and the features of the model. It also describes the execution and sequence flow of the main components of the model that would be developed. Therefore, the modeller must describe the sequence in which the activities follow each other. In addition to that, modeller ensures that the described sequences are structurally equivalent to the sequence of business functions as described in the organizational workflow.
Therefore, the different programming languages for web service orchestration can be used to describe the sequence of which the services are called and make the sequence executable (Coalition, 2012). In addition, business process modelling standards such as BPMN and BPEL that are based on orchestration languages have to be evolved for supporting the conversion from business process model to the executable service orchestration.

Consequently, BPMN used as a methodology to design, analyse and implement our process model workflow does not allow call of geo-services through its element. Although, chaining geo-processes require defining the sequence of which tasks or activities that are used to perform a single process (processing unit) or combined processes in more complex structures. A high-level orchestration can be defined as a container for individual processes activities and workflow within (Strickland et al., 2006).

In fact, BPMN is an XML-based modelling tool that specifies how to define a business process in terms of orchestration or chaining of existing web services(Database and Expert Systems Applications: 17th International Conference, DEXA 2006, Krakow, Poland, September 4-8, 2006, Proceedings, 2006). Service chaining is a process whereby, the output of the first called service in the workflow can cover at least one input to the next service. In the geospatial process, chaining services are needed when no single service can fulfil the business goal. International Organization for Standardization, (2005) and Rob Lemmens et al. (2007) distinguishes three approaches of chaining geo-services:

User defined chaining: In this chaining type, the human user defines and controls the order of individual service execution (Rob Lemmens et al., 2007). Hence, it is the role of the user to discover and to evaluate the fitness of services by determining a valid sequence of service and controls the execution order. Additionally, users are responsible for ensuring that the inputs and outputs of individual services are compatible and the complete chain is semantically correct.
This pattern is also called transparent chaining because the details are not hidden from the users as explained in Schaeffer, (2009).

Workflow Managed Chaining: in this approach a human user invokes a service that controls and manages the chain; also that service is aware of the individual service in the chain. The involvement of the user in the step of this type of chain is mostly watching the individual service execution (International Organization for Standardization, 2005). Its difference to the previous type is that the existence of a defined chain is assumed prior to the user executing the chain Schaeffer (2009) and Rob Lemmens et al. (2007).
In addition, the user can provide the parameter particularly to a specific instance but, relies on workflow service to carry out the chain. In this pattern, the chain is abstractly predefined and stored in the workflow engine. Based on predefined service chain, the workflow service is able to determine what are the appropriate data source, processing services, control the sequence of execution and present the final result to the users (Schaeffer, 2009)

Opaque Chaining: In this pattern, all participant service appears as a single service to the user (International Organization for Standardization, 2005). It has a central control service which handles all coordination of the individual services that are part of the chain and hide all details to the user. The user may or may not be aware of the fact that the aggregate service hides a chain (Schaeffer, 2009). Therefore, the aggregate service is responsible for service coordination. Like the workflow managed pattern, the existence of predefined service chain in the sense of deployed instance of the chain is assumed(International Organization for Standardization, 2005 and Rob Lemmens et al. ,2007).

In this research, we used opaque chaining pattern because all participant services in workflow appear as single service to the user. In addition, in the approach we used BPMN has process engine which controls the execution of the workflow. That means process engine can work as a service which controls individual service execution as it has to delegate service task to execute in the workflow. From literature review the following requirements are captured to model workflow activities in terms of orchestration where many of them are provided by BPMN itself:

- Identify all elements required for process model to chain service. BPMN present service chaining in the form of diagrams.
- After identifying the elements, the next step is to model the process using the process model editor. The modeller needs to spend a lot of time here by specifying how tasks are connected to each other, the sequence flow and their order of execution.
- Define the process data. In this step, the modeller introduces the data required by the process model. Therefore, he needs to specify modelling entity relationship diagram and the capability to use different data format like XML etc.
- Control flow structure needs to be considered to allow direct implementation of service chain,
- The mechanism to allow the hierarchical decomposition as key feature of orchestration must be considered for dealing with complexity of process workflow,
- Data handling mechanism is needed for defining, transferring and manipulating the complexity of data structures,
- Exception handling mechanism must be defined because the execution in orchestration deals with several operations which require taking into account the error handling mechanism during the execution,
- Mechanism of message correlation must be considered because; during the execution the multiple orchestration instances run in parallel. Then, to support the correct routing of the message the modeller must define how messages are correlated.

- Mark all BPMN element representing geoprocessing services in BPMN diagram
- Create a special element with clear semantics to identify BPMN elements representing geoprocessing services in the process model.
- Define a mechanism to send service execution request to WPS Server.

3.3 Moving from high-level orchestration to low-level service implementation details

In fact, here we use three different process design approaches to model geo-processes. The modeller may use them in their natural separated order. The first approach is abstract process specifications which define the complex tasks in an abstract way. This approach is suitable for describing the processes at high-level abstraction and makes the description more suitable for all kinds of users (Barker & Van Hemert, 2007).
The second approach is considered as an intermediate approach which expresses the power of the complete notation to describe activity flow and the exception paths (Barker & Van Hemert, 2007). At this step, the model still not executable but all semantics behind the tasks are defined and subjected to the validation rules. Moreover, the modeller can define some pre-conditions, post-conditions and select instances of the processes which participate in orchestration(Barker & Van Hemert, 2007). Even if some technical details such as specification of data structures required are omitted at this step but orchestration can be executed to test the validity and completeness of process model(Barker & Van Hemert, 2007). This approach corresponds to an algorithmic description of a conceptual model (Barker & Van Hemert, 2007).
The last approach is execution which makes possible to execute orchestration model specification by using process execution engine. Today, BPMN is the most adopted for modelling business processes as it is a graphical notation with a clear semantics. In addition to that, it represents the mechanism to obtain semi-automatic XML codes to deploy in orchestration engine (Barker & Van Hemert, 2007).

After analysing our orchestration requirement and what BPMN can offer, we can use this standard in this research to create a workflow containing web processing services. As we said before BPMN offers notation elements to represent services at high-level. Therefore, lower level description of the workflow is needed to make it executable. Although, the main solution adopted for executing BPMN models up to version 1.2 was through their mapping to another language. Moreover, serialize BPMN diagram using both XML Process Definition Language XPDL (Coalition, 2012) and BPEL (Jordan & Alves, 2007b), a block-oriented language is possible. Serialization is the process of translating data structure from one format to another format that can be stored and reconstructed in the same or different computer (Shapiro et al., 2012).

In Ouyang, Aalst, Wil, & Arthur (2009) an algorithm can be found for automatic transformation of business process diagram components to a block-structured language BPEL. But, the current version (2.0) of BPMN (Object Management Group et al., 2011) brought important changes and interesting innovations. The most important innovation in this current standard is that BPMN models can be stored in a standardized XML-based format and the introduction of Metamodel (Hallwyl, Henglein, & Hildebrandt, 2010).

Based on these new features BPMN 2.0 models can be exchanged between tools and direct execution of BPMN 2.0 is possible. Therefore, mapping to another language for execution is no longer needed according to Hallwyl, Henglein, & Hildebrandt (2010) and Allweyer (2010). BPMN interact with external systems by using so-called service task, consequently, no BPMN element or connector was developed by OMG in BPMN for calling OGC services. The service task provided by BPMN is only for SOAP web services which flow WSDL specifications for allowing interoperability between systems.
Therefore, in this project, a new mechanism of calling spatial services in BPMN model is implemented by developing API and an external python script.

This API identifies BPMN tasks representing geo-service activity in the workflow, read their properties and make service call through a python script. Though, in this API tasks that represent geo-services in the process model, are identified and all its properties are known for making service call. The low-level service description is implemented in this API interface and it triggers execution of python script to send a service request to WPS server. Moreover, input parameters of the process model are supplied by the user through a developed web entry.

3.4 Procedures of Service Chaining with BPMN

Today's research on application and earth system involves the acquisition, analysing and modelling heterogeneous geospatial data (Díaz, Pepe, Granell, Carrara, & Rampini, 2010). Where geoprocessing and spatial analysis are becoming more complex and require a combination of multiple services. In addition, some spatial questions involve actors, activities, resources, and result to be modelled. Therefore, "service chain is a solution aimed to solve complex application tasks by changing the way of development and deployment of those applications" Zhao et al., (2012) and Friis-Christensen, Ostländer, Lutz, & Bernard, (2007). As a result, "by wrapping data and processes with web services it is easy to transform a spatial processing model into a service chain" (Meek, Jackson, & Leibovici, 2016).

Generally, a geo-service process can be an atomic process which runs independently and composite service (orchestration of service) which consist of a sequence of processes in a predefined pattern. Orchestration of web processing services is a composite process. Because several web services or geo-services are chained to solve a spatial problem. A process is defined by its inputs, outputs and operations. The relationship between activities in process model is modelled by specific symbols of Business Process Modelling and Notation (BPMN). Furthermore, the input of each task or activities in process model is specified based on the operation. For example in computation buffer, a script task requires two inputs parameters: Distance and data input specifying the required data for specific operation. For example URL of a GetFeature request for WFS which is delivered as input for executing the request.

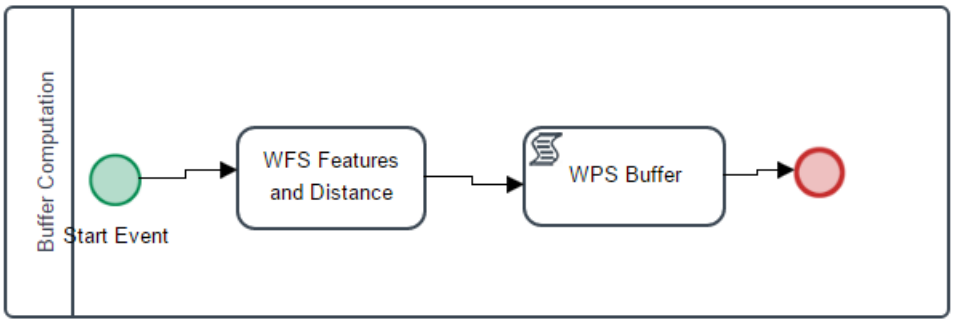


Figure 3.1: Example modelling buffer WPS operation with BPMN

Figure 3.1 shows the required inputs parameters in the process model of computing buffer. In this process, two parameters are required to be provided by the external user in the process: Distance and WFS URL indicating the location of the spatial data required for that operation.
In fact, BPMN does not provide task or connector for calling an OGC services such WFS, WPS etc. The way of handling this problem is discussed in chapter four.

In service chaining process, it is required to specify control structure like sequences and the conditional statements (If-Then-Else) indicating what task is coming after another. The important feature in service chain is to specify how the output of one operation can be used as an input to another operation in process model based on organizational workflow. From literature review we identified different procedures for combining multiple services into service chains:

- Identify all BPMN elements required for modelling processes based on organizational workflow. BPMN present service chaining in the form of diagrams.
- After identifying the elements, the next step is to model the process using the process model editor, for example, ProcessMaker.
- Mark all BPMN service task representing geo-service in BPMN diagram, and any other additional settings. Tasks in BPMN diagram can be assigned to the different job; in this case task representing geo-service in the diagram was mark as WPS service so that the API will be able to distinguish them from the other tasks during the execution of the workflow.
- Each task in the process model should have a unique number to differentiate it with other tasks. The tasks in process model are equipped with a unique identifier, this identifier helps process engine to delegate task for execution by reading its position in the process model.
- Create BPMN element able to identify those element marked as geoservices to allow them to make service call. In addition to that, this element should allow the interaction between different components involved in our proposed method and also allows saving the service execution result to model variables repository.
- The service requester should be able to find out about the process status and, in case the process has completed, where to pick the result. For single atomic WPS service execution, the user sends a request and waits for a response after service execution. In case multiple services need to be combined to answer user questions, a methodology of combining them is needed and the involved services need to be identified in the workflow. The services are modelled as activities into the workflow so that a single request of the user executes workflow into a service chain. During workflow execution user can still receiving the information about chain execution and he can also be able to see where he is in the workflow.
- All involved geo-services were described based on OGC standard. In WPS implementation, it is necessary to know how to build a request for one or more operation for its description or execution because WPS is accessed from the web browser by the use of Hypertext Transfer Protocol (HTTP), which enables communication between client and server. This protocol uses two methods Get and Post for accessing WPS Server. HTTP Get is used for retrieving the targeted information from WPS server and HTTP Post is used to submit the data to be processed to specified resources.
- Develop an external python script for sending WPS execute post request and receiving a response from the WPS server. After modelling activities and their sequences of execution, identifying the geo-services and their properties in workflow, describing the services based on standard and needed operations, the last step is to send execute post request to WPS server. A python script was developed to send execute post request to WPS server for execution (see table 2.4).

3.5 Summary

In this chapter, different approaches of orchestrating web processing services (WPS) with BPMN have been identified. In addition, the procedures and the functional requirements of chaining two or more web services or geo-services were discussed. We have also identified the requirements of moving from high-level service orchestration to low level service implementation details. As a result, two or more services are chainable if an output of the first called service can cover at least one input of the next service. In BPMN, the services must be flowed sequentially and defined in the way that an output of the first service is an input to the next service. The defined requirements in this chapter are the inputs to the next chapter as it will help to design a method to orchestrate two or more services with BPMN notation.

4. CHAINING WPS PROCESSES WITH BPMN

4.1 Introduction

The objective of this chapter is to propose a method of extending the functionality of Business Process Modelling and Notation (BPMN) to include geoprocessing services. The proposed method will allow BPMN to model geospatial processes by developing a mechanism of how BPMN element representing geospatial services in the workflow can be identified and adapted to make a service call through a developed API and python script. The chapter starts with introducing process modelling with BPMN in section 4.2, Extending BPMN functionality in section 4.3, Components of new proposed method in section 4.4 and the last section 4.5 is a summary.

4.2 Process Modelling with BPMN

BPMN specification presents the way of obtaining automatic XML code to deploy in workflow engines or to be shared in different domains. Therefore, process modelling with BPMN can support different methodologies to represent the workflow and the participants as well as different modelling goals such as orchestration using the actual business process (Chinosi & Trombetta, 2012). As discussed before, several GI questions may require the combination of multiple services where the output of one service is an input to the next service (See Figure 4.1).
In figure 4.1 for example, user need a method of combining the listed dataset with corresponding operations so that he can get the final result from the operation combining all dataset required from a single request. Therefore, the requirement listed in sections 3.4 can be applied in this case to define a method whereby, based on organizational workflow BPMN can be used to model geo-processes, their corresponding inputs and outputs.

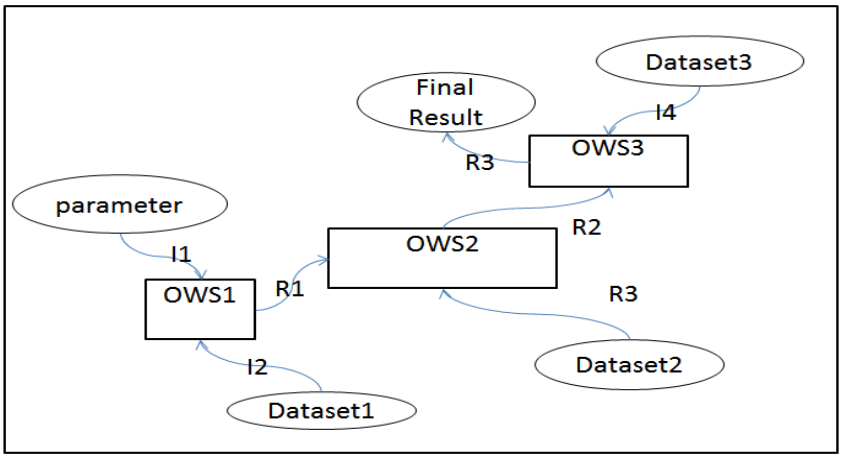


Figure 4.1: Example of complex question that users may have

Legend

I: Input
R: Result or output

OWS: OGC web Services

: Process

: Input dataset

In BPMN we count three levels of process modelling:

- **Process Mapping**

Process mapping is the first activities within BPMN to model processes, in section 3.4 we listed some procedures of chaining spatial services within this notation to include spatial services. The first two procedures concern with process mapping where the modeller, after analysing workflow of activities he can choose the BPMN elements to use and an editor which provides BPMN notations. As we said before here, the modeller needs to spend a lot of time by specifying how tasks are connected to each other, the sequence flow and their order of execution based on organizational workflow. The result of this level is a chain of activities in the form of BPMN diagram.

- **Process Description**

Given that, all processing units services and their inputs was modelled using BPMN notation, at this level process is extended with enough information, like marking the tasks representing geo-services, create variables to contain inputs and output from processing units execution, create dynamic form and step, create users and process rules, and assign tasks to corresponding users.

- **Process Model**

This is what in our case we call chain of activity or BPMN diagram, at this level process model, is equipped with enough information which makes it ready for being analysed and executed. ProcessMaker used as BPMN editor that has an essential element to execute the workflow, but is not able to distinguish, activities representing geo-services to make service call as it was not designed for that. Therefore, this research we proposed the creation of special element able to identify all tasks representing geo-service activity in workflow and read their properties to make service. This element should be equipped with enough function to allow connection of BPMN tasks representing geoservices in BPMN diagram (workflow) to their corresponding services make service. The involved WPS services are described based on OCG WPS execute post request as explained in section 2.4 (see table 2.4).

In section 4.3, a method to model geo-services with BPMN is designed based on the requirement. This method will answer the above questions (see figure 4.1), whereby, the processes, inputs and outputs represented in the workflow are modelled with BPMN into BPMN diagram. Figure 2.5 explains the basic BPMN elements to create BPMN diagram. To that end, the procedures listed in section 3.4 are flowed to include web processing services in the diagram, remember that BPMN present service chaining in the form of diagrams.

4.3 Extending BPMN Functionality

As explained in problem statement of this research, and section 4.2 most of the users require the use of multiple services to respond to GI questions, it may happen that some questions require the involvement of actors, activities, resources, objectives, and outputs in solving such questions. Moreover, an architectural design is needed to allow service combination for satisfying one or more needs.
In order to respond to the spatial question and real-world processes which require the combination of multiple services and data from different sources, we should consider several steps to model geo-processes. This research comes up with a new method of modelling geo-processes with BPMN which was not natively created for modelling geo-services (see figure 4.2). The different research was conducted on how BPMN can be used in the realization of a complex process model that involves several WPS instances to be chained to one another, to define a geoprocessing workflow.

In Schaeffer (2009), OGC has been working on the integration of their standards with orchestration languages and standard toward web service orchestration. They have explored the use of BPMN as an option to create spatial web service chaining. In Prager, Klímek, & Růžička (2009), BPMN technology has been used to enable integration between any spatial services. Moreover, Albrecht, Derman, & Ramasubramanian (2008) introduces the use of ontologies and model such as BPMN to standardize the way that process communicate to each other in geospatial languages.
This research describes a system that supports geospatial requirement into a business process. In addition, those requirements can be integrated into BPMN business workflow by marking BPMN elements that represent spatial service in the process model and develop a new element able to identify and read their properties in the workflow to make a service call (see Figure 4.2 and 4.3).

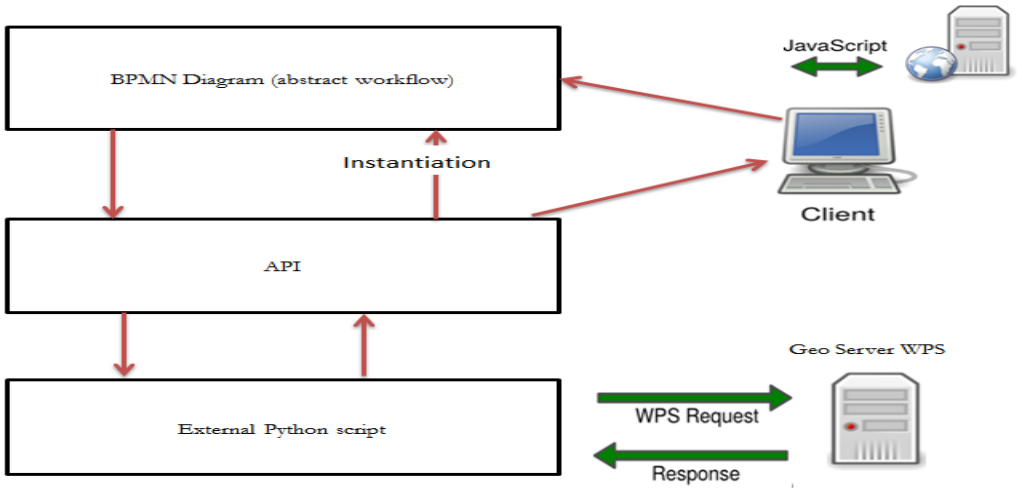


Figure 4.2: Method for extending BPMN to chain geo-processes services

Figure 4.2 illustrates the main steps of proposed method to extend BPMN to model geo-processes. Within this method, two or more executable WPS processing units instances can be chained in one-way or another within BPMN diagram to answer GI questions. The steps within this method are described below:

- **BPMN Diagram**

BPMN was used in this research to define a sequence of activities and their execution order based on organizational workflow. Consequently, it was not designed to model geo-processes as explained in problem statement of this research. For this diagram to include web processing services, we developed an able to identify all BPMN elements marked as geo-service in the workflow to make service call.

- **API**

API is a special element developed in this research based on the requirement to allow BPMN to include geoprocessing services. This element is able to identify all BPMN tasks marked as geo-services, once those tasks are identified; it triggers the service parameters in right order to make a geo-service call. However, within this API, the inputs parameters of geo-services in the workflow can be queried from workflow model repository (Database). It also defines the necessary functions to interact with all the components of the method.

- **Python Script**

This script was developed to send service execute request to WPS server. Once API has identified tasks representing geo-services and read their properties, the next step is to trigger the execution of python script. This script takes inputs of service to be executed from the API as the API is able to directly communicate with BPMN workflow when the workflow is running. In section 2.4 of this thesis, we explained how this request is sent to WPS Server.

- **Client**

The client in this context is a user, how managed the workflow. The role of this user is to provide the required inputs initiate the workflow. This user will get the status of workflow execution until he gets the final result on the web page.

- **Geo-server WPS**

Geo-server is an open source server designed for sharing spatial data. This server allows flexibility in map creation and spatial data sharing, in addition to that, it also publishes data from any spatial data source over open standard. WPS is not a part of geo-server by default; it is only available as an extension. Geoserver within this extension acts as WPS implementation server.

In fact, within this method, WPS operation as a service can be represented by a task marked geo-service in BPMN workflow to represent geo-process in the workflow at high-level. In BPMN workflow modelling, web service can represent one or more activities (tasks) to carry out the business process in the process model. Thus, orchestration can be defined as sequence flow of all activities or services involved to carry out the business process in the organizational workflow. As discussed before BPMN service task is used to represent the spatial service in the workflow but it call still work as normal BPMN service task for none geo-processes tasks.

4.4 Components of new proposed method

Figure 4.3 illustrates the functional and technological architecture of the main components of proposed method for extending BPMN to include geoprocessing services. The structure is made up of four components. The role of each component in figure 4.3 is explained below:

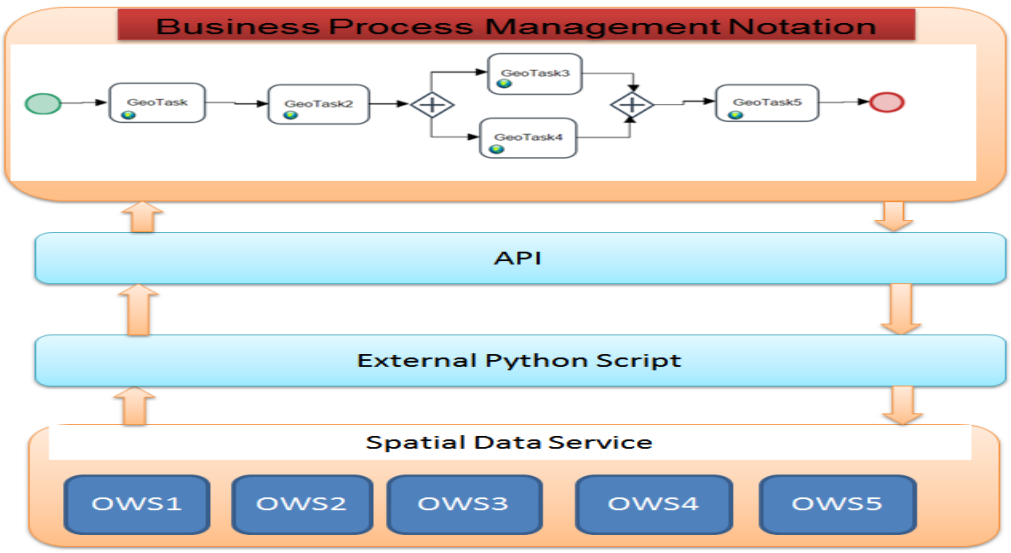


Figure 4.3: The main components showing how BPMN can be extended to handle WPS processes

4.4.1 BPMN Diagram

The first component in this technological architecture of chaining WPS services using BPMN is a BPMN diagram (workflow). This workflow is a set of user activities into complex tasks, it is normally suitable for defining activities at high-level. During this step, the modeller needs to identify BPMN elements to describe the activity flow and define their sequence of execution. In addition to that, the modeller needs to add all required information such as variables, dynamic forms, steps, business rules and other settings to make workflow executable. Remember that BPMN was not natively created for handling geoprocessing services. Therefore, all tasks representing geo-services in the workflow are identified and marked to be differentiated with the other tasks in the workflow. Then, during workflow execution, the delegated task needs to be checked and linked to their properties to make service call.

4.4.2 API

API is the most important element in this architecture as it makes BPMN chain geoprocessing services by identifying BPMN tasks representing geo-service to make service call. This component has the necessary functions to communicate with the workflow during its execution. As explained in previous components, the user draws a workflow and put all addition settings to make it executable. The result from BPMN is a diagram showing the sequences of activities in the workflow. Workflow can contain the geo-processes and none geo-processes.

Figure 4.4: Workflow showing how a task service can be used in a different way in workflow

In this workflow, service task is used to represent geo-service but it can still use in its normal way during the model execution. Developed API is only concerned to Geo-service; by convention, a service task representing geo-services in the workflow should start by the abbreviation of the service name that it represents such WPS, WFS, and WMS etc.

In the above example service task representing geo-service calls feature about the urban area and next service task calls a web service function that calculates the population density. The role of this API is to identify what are the tasks representing geo-processes in the workflow and equip them with enough information to make service call. Once those tasks are known the API reads their properties such as inputs parameters from the model variable repository and triggers the execution of python script to set service execute request WPS server. Once this execution is completed the script notifies the API the completeness or the status of execution. API also notifies the completeness of geo-services execution to process engine to continue executing the rest of the workflow up to the end. This operation will be repeated up to the end of workflow execution. The result of this execution is saved in the folder created on the computer and its URL is sent to model variables repository. The following are the required functions for this API to perform his job:

a. CheckTaskType()

This function checks the type of delegated task by querying BPMN_activity table of workflow schemas or by using Processmaker REST API. Then, if the type of task is service task the function *InstantiateTask()* is called to check if the task is representing geo-service activity in the workflow. In fact, when the execution of workflow starts the API is triggered and process engine creates CaseID of the running process and delegation index on delegated task to run, that index is sent to the API. By calling *CheckTaskType()* function the API is able to know the type of task. If the task delegated is service task API *InstantiateTask()* function is called otherwise another function is called.

b. InstantiateTask()

This function check if a returned task by the previous function is representing geo-service activity in workflow by reading its properties in process model repository. Once the function is presenting geo-service, an instance of that task is created to make service call and *getVariables()* is called to return an array of parameters (variable names and values) attached to that task.

c. GetVariables()

This function can use either Processmaker web service or query process model database to get variables and values linked to that particular task. The returned array of variables and values are saved in API global variables to be used later in *ExecuteService()* function.

d. ExecuteService()

This function triggers the execution of python script with parameters returned *GetVariables()* function. *ExecuteService()* function triggers the execution of python script. After the script is executed it inbox the API by sending the execution status if the status shows the successful completion of the service execution API *SendResult()* function is called to send the result to the model variable repository.

e. SendResult()

This function is called after python script is successful executed, it is used to send the URL of the service execution result to the model variable to be saved and used as an input to the next activity of the workflow.

f. NextTask()

This function is called to handover activity to process engine and route case to the next task in the workflow.

4.4.3 External Python Script

The role of this component is to send WPS execute post request to WPS server. This script gets parameters as system variables from API, those parameters are URLs of WFS GetFeature request or any other parameter depending upon the operation to be executed. Post request sent to WPS server requires three parameters:

- URL is the address of WPS server where geo-processes are deployed and implemented,
- Data is an XML file specifying which WPS operation to be executed, inputs parameters, data types and format of the result,
- The last parameter is header which defines the content type

In section 2.4 we described how WPS execute post request is sent to the WPS server for executing one or more service requested. In table 2.4 we provided an example on how WPS execute request is sent to the server.

4.4.4 OGC Web Services

OGC web services are cloud services provided by OGC to allow users to integrate spatial data in their application.
Figure 4.5 shows how those components sequentially interact with each other during the execution of the workflow.

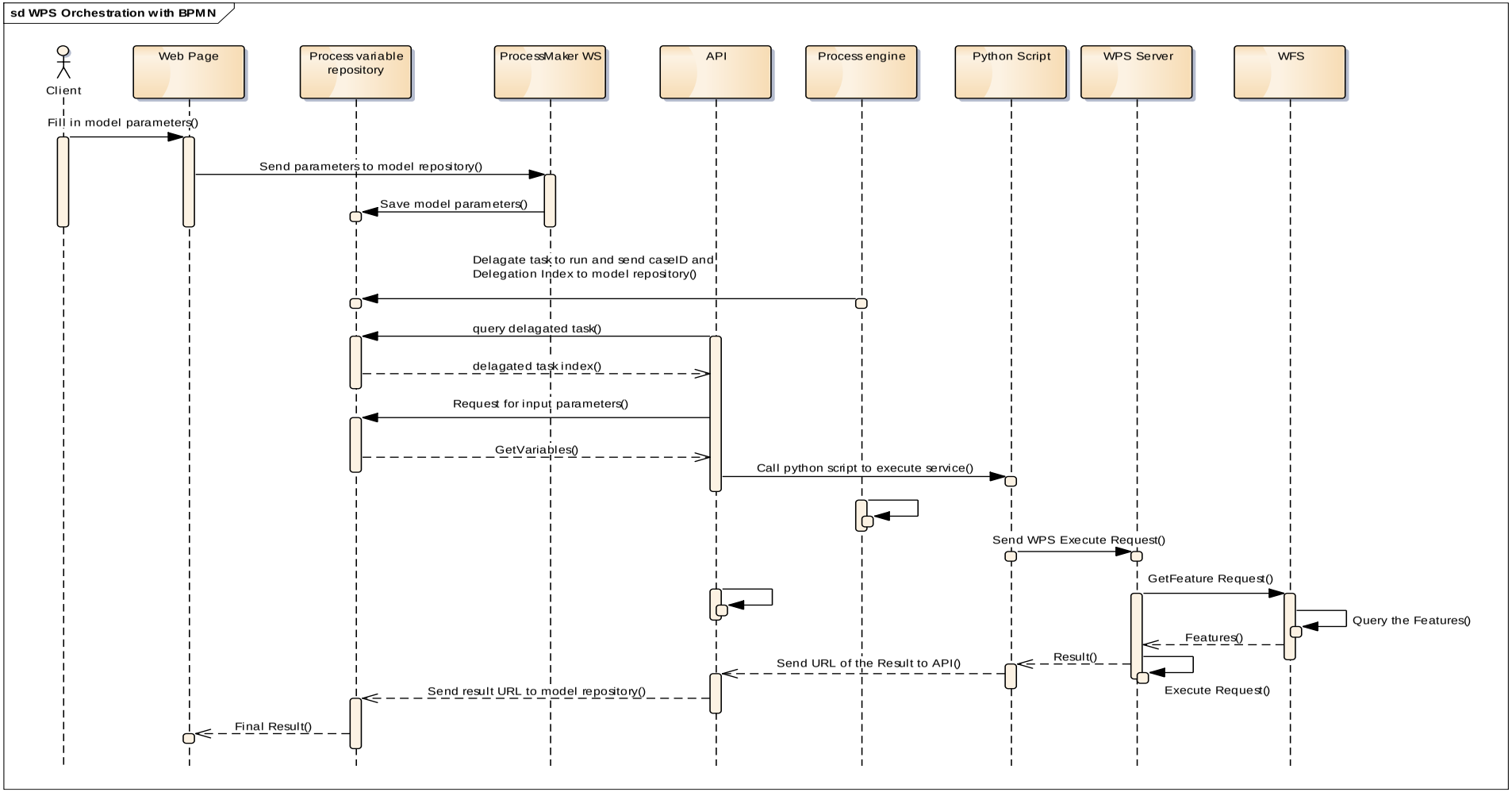


Figure 4.5: Sequence Diagram showing the interaction of components involved in WPS chaining with BPMN

The API functions are called in an iterative way to do their job in the workflow. Figure 4.5 capture the flowchart showing which API function is needed and when it will be called during the execution of the workflow.

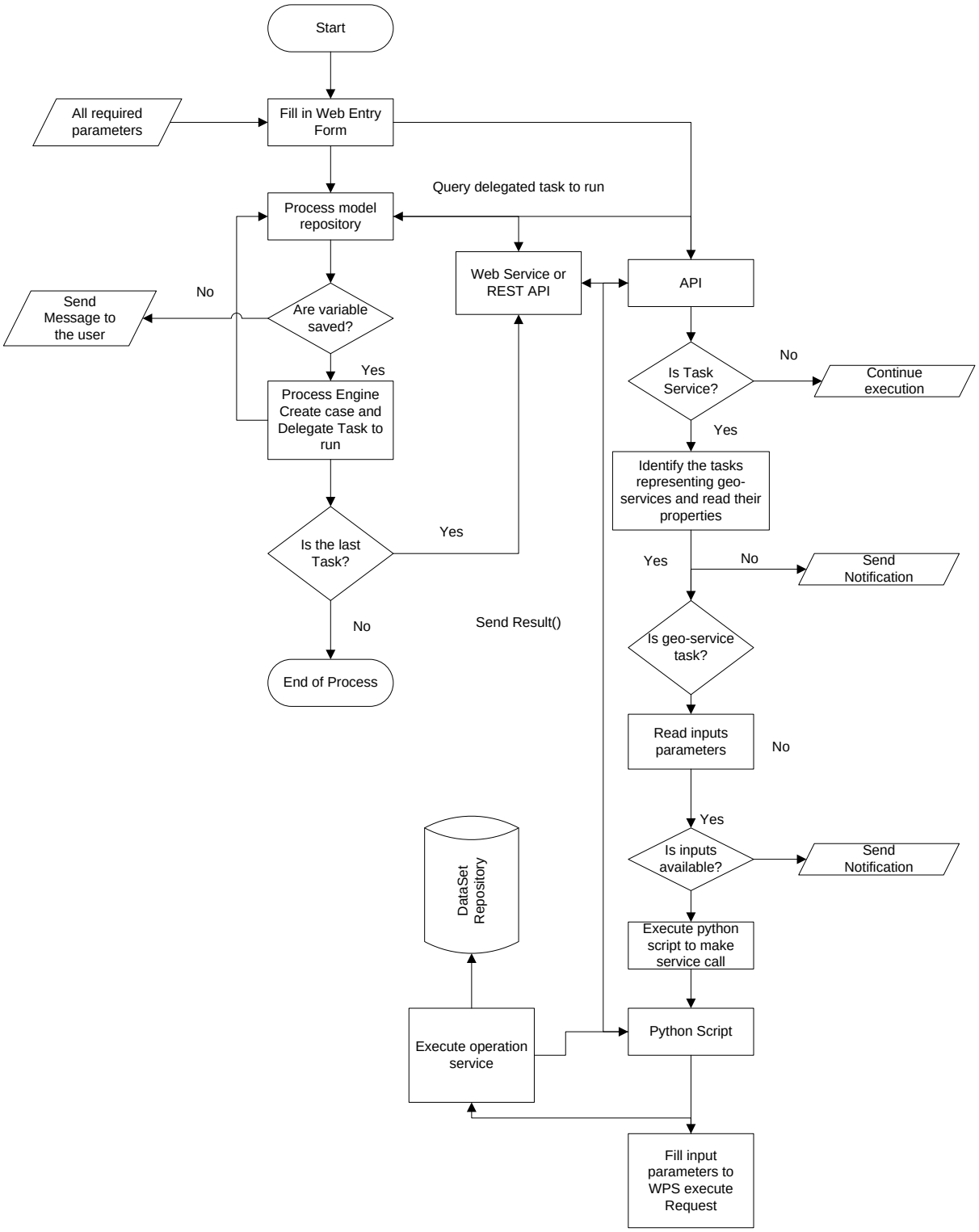


Figure 4.6: Flowchart showing the execution steps of BPM diagram, API and python script

WPS returns a response which is an XML-encoded to python script and saves it to user created folder and file. The URL link corresponding to this result is sent to model variable repository through the API *SendResult()* function. In this research, we used Geo-Server WPS extension for geoprocessing functionality on Apache Tomcat. The result of the execution can then be transmitted to the next activity in the workflow as input parameter which defines orchestration. This process will be repeated until the end of workflow execution.
To implement this method different technology, tools and programming languages are used. See table 4.1

Components	Technology, Tool and programming language
Process model or workflow	ProcessMaker
API	PHP
External Script	Python
OWS	Cloud services, Geo-server

Table 4.1: Technology, Tool and programming language used

ProcessMaker: is an open source workflow management system. It is BPMN implementation; it has all necessary BPMN elements for a small organization to create their business workflow. ProcessMaker has a database for storing process tasks and all setting related to it.

PHP: PHP is server side programming language used to create API to allow remote access to the services, workflow functionality, identify BPMN element representing geo-service, and make service call services through a python script.

Python: Python was used to create a script that sends WPS execute post request to WPS server as requested by Process engine through API interface. WSDL Web Service Description Language as an XML-based language was used to describe and locate web services. In addition to that, three client-side scripting languages (Open Layer JavaScript extension, HTML and CSS) were used to implement the user interface.
Geo-server: is used as WPS implementation server. The spatial services are described based on OGC standard to allow them to be discovered from the web.

4.5 Summary

In this chapter, a new method for extending BPMN to handle WPS processes was proposed as a solution for an organization to include geoprocessing services in their workflow and model to geo-processes that require the combination of multiple services. The different component of the proposed method was explained in this chapter. BPMN was used to provide the basic notation to create a sequences and execution order of activities in the workflow. An API was created to identify all BPMN tasks representing geo-services in workflow and make service call through a developed python script. The result from this call is saved and its location is sent to process variable through API function to be used later in the rest of workflow execution. The script defines post request specifying which operation to execute, server address content type etc. this method is implemented in chapter 5 to demonstrate the functionality of newly developed method.

5 IMPLEMENTATION OF WPS ORCHESTRATION METHOD

5.1 Introduction

For the implementation of new Landuse plan, the BPMN is used to design workflows containing activities including their logic sequence relying on structure proposed in chapter four. BPMN is also used for describing the workflow at information viewpoint and its composition into single activities. As discussed modelling spatial processes with BPMN needs a special element as it was not designed for interacting with spatial data and geoprocessing services. Moreover, with BPMN editors such as Processmaker, it is possible to configure the inputs and its associated activities using technology tool and programming to implement each task in the workflow. An example of such functionality is used to illustrate how BPMN can be extended to orchestrate spatial services such as WFS and WPS in land use planning. The chapter starts with System prototype functionality in section 5.2, and summary in section 5.3.

5.2 System prototype Functionality

The aim of this prototype implementation is to test that the method for extending the functionality of the BPMN standard to model geo-processes and call spatial services works. The chosen scenario of this research is to help the planner of Enschede municipality to know which part of the urban area that will be affected by the expansion of forest. In fact, in land use planning activities knowing the land use of a given area and its boundaries is one of the most activities in geo-information application. Therefore, the municipality of Enschede is planning to update their land use by only expanding the forest area.

Three required datasets are available as web service but the way of combining them based on organizational workflow is needed to identify the part of the urban area that can be affected by the expansion of forest in new landuse plan. Here, the planner is only interested to know which part of the urban area affected by new land use planning implementation which takes care in expanding the environmental area such as a forest.

This case was chosen as proof-of-concept because it involves different geo-processes which need to be chained in sequence way following the organizational workflow such that the result of the first executed service is used as an input to the next operation or activity and the rest of workflow execution. Figure 5.1 depicts, as workflow various spatial operations and its inputs of our chosen scenario. The figure also shows how the tasks are sequentially defined in the view of the human to produce the result. Additionally, figure 5.1 also show the relation between operations and their inputs.
This scenario requires five inputs where three of them are WFS GetFeature URL which queries features from PostGIS database, buffer distance and the address of WPS implementation server.

WFS GetFeature URL inputs:

- http://win371.ad.utwente.nl/cgi-bin/mapserv.exe?map=//win371.ad.utwente.nl/student/s6019978/Thesis/configWFS.map&SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=neighbourhood
- http://win371.ad.utwente.nl/cgi-bin/mapserv.exe?map=//win371.ad.utwente.nl/student/s6019978/Thesis/configWFS.map&SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=forest_areas
- http://win371.ad.utwente.nl/cgi-bin/mapserv.exe?map=//win371.ad.utwente.nl/student/s6019978/Thesis/configWFS.map&SERVICE=WFS&VERSION=1.0.0&REQUEST=GetFeature&TYPENAME=urban_areas

WPS implementation Server Address:

- http://130.89.236.184:8080/geoserver/ows

The process that planner carries out, in this case, is as flows: First, the administrative boundary of Enschede is required and expanded at a given distance. Second, the result from expanded boundary is intersected to forest dataset. Finally, the result from this intersection is expanded at a certain distance and intersects with an urban area in order to get the urban area that affected by the expansion of forest area.

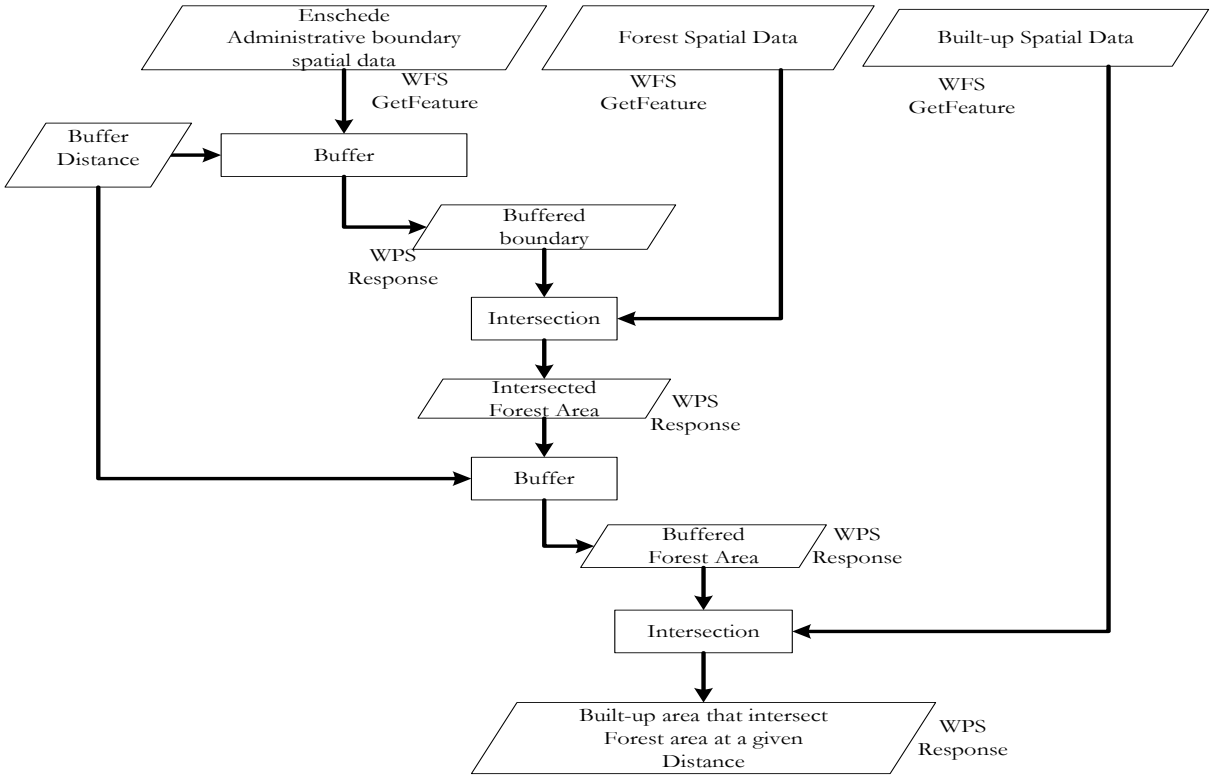


Figure 5.1: Organizational workflow of the activities in the office of planner represented as flowchart

5.2.1 Modelling landuse planning processes with BPMN

The first need for the prototype is to have the chosen process, as depicted in figure 5.1, modelled according to our approach. To that end, the execution order of activities and their sequence flow as illustrated in figure 5.1 processing services and their inputs were modelled using BPMN notation (see Figure 5.2).

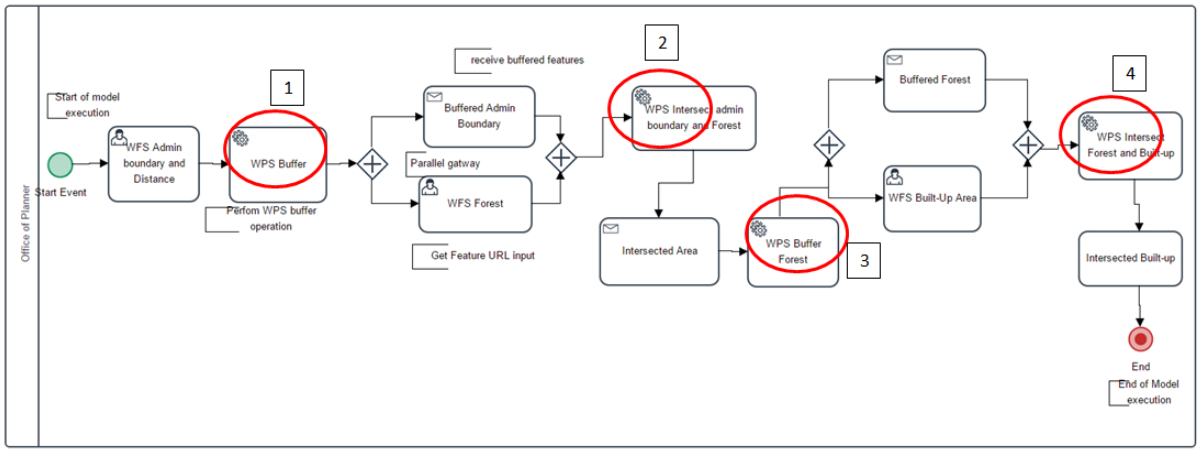


Figure 5.2: BPMN Process model for land use planning model.

For modelling, the landuse planning activities different types of BPMN elements were used. BPMN Service task (See figure 2.5) is used to represent geo-processes service in workflow, BPMN Receive task (See figure 2.5) is used to show in workflow the activity that receives the result from the previous processing unit and finally, a BPMN user task (See Figure 2.5) is used to represent tasks that require user inputs see figure 2.5 of chapter two for more explanations about used task types and other used BPMN elements.
As explained in previous chapters this standard is not designed to interact with spatial services. To call spatial services within BPMN, service task have been used as a task that represents geo-process service in the landuse planning workflow. Moreover, tasks that correspond to geo-service in process model were adapted to make a service call through a developed API and python script.

In Figure 5.2 service tasks are marked with a red circle to show activity representing geo-service in land use planning workflow. The marked task is numbered as shown in figure 5.2. (1) WPS Buffer this is the first operation of this workflow which creates a buffer on the administrative boundary of Enschede. (2) WPS intersect admin boundary and forest this operation intersect both dataset to have forest are that intersect the boundary of Enschede. (3)WPS buffer forest, this operation create a buffer on the forest that intersects Enschede administrative boundary. (4) WPS intersect forest and built up this is the last operation in this workflow it shows the final result of the area affected by the expansion of the forest. In addition, each task in the workflow is assigned to the user in charge, in this case, all tasks are assigned to the planner. The second step after mapping all organizational processes in BPMN workflow, marking different activities in workflow and assigning activities to the corresponding user is to create workflow variables to store process data.

Creation of process variables

Process variables are defined as variables created within the project to store process data. They can be used in any of the objects in the process. These variables are also used to populate fields in dynamic forms within the process model. For landuse planning workflow, different variables were created to store process data during workflow execution. To provide values to these variables dynamic form was created and the filed in dynamic form were linked to their corresponding variables.

5.2.2 Landuse planning model realization

A web entry was created as a mechanism for the planner to pass on the workflow details about the datasets (services) to be used and any other parameters required for the execution of the workflow. In BPMN designer, web entry is linked to specific start event which indicates where it can initiate the workflow (see figure 5.3).

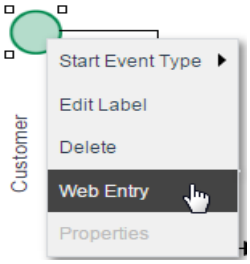


Figure 5.3: Create a web entry on start event to initiate workflow

Web entry provides a web link to dynamic form in the first task of the workflow (see figure 22)

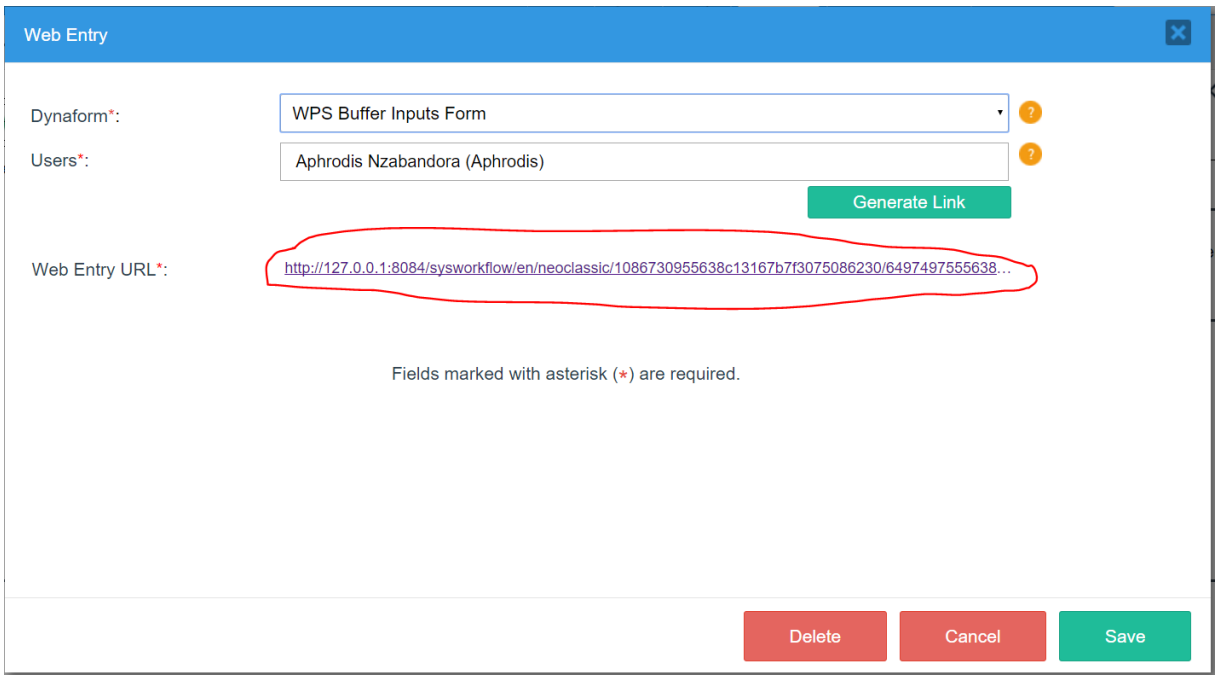


Figure 5.4: Example of how Web entry is linked to dynaForm in workflow

The role of this web entry is to enable the planner to initiate a new case with all required inputs from the external web interface without manual logging to ProcessMaker. For the planner to externally login ProcessMaker, ProcessMaker web service login function was called in web entry to provide access to external users figure 2.2 shows how web service is discovered by the service requester. Within web entry, the planner fills out the dynamic form (see figure 5.5) and submits data to the workflow process data repository through the web service. After inputs submission process engine initiates the workflow by creating CaseID and delegation index of the task delegated to run, this index is sent to API.

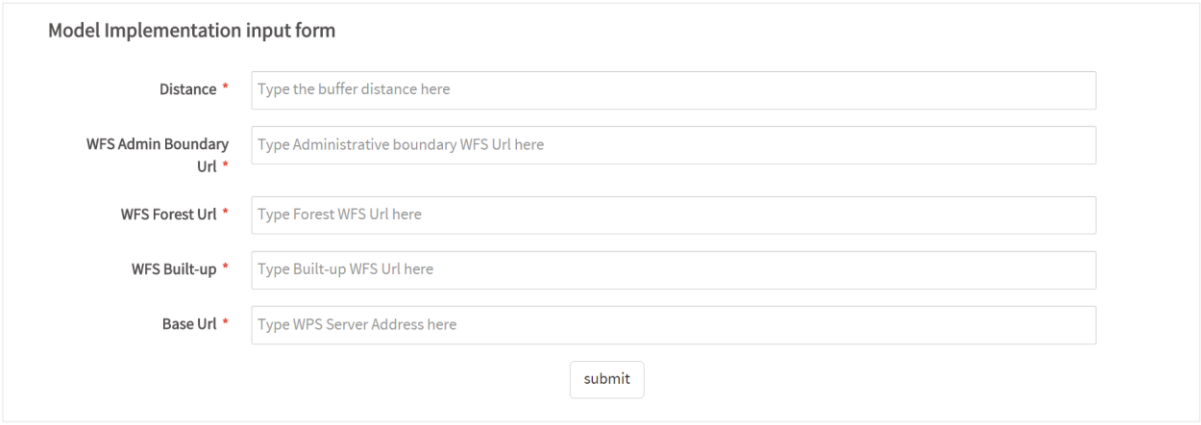


Figure 5.5: Landuse planning workflow parameter inputs

This interface allows the user to provide input parameters to the workflow variable repository through web entry and web service. The landuse planning workflow requires five inputs parameter: WFS GetFeature URL of Enschede administrative boundary, Buffer distance, WFS GetFeature URL of Forest area, WFS GetFeature URL of the urban area and Base URL which is an address of WPS server. The interface that displays the result to the user is developed using internet browser as application software that is able to access a web server and interpret client codes (HTML, Open Layer JavaScript extension and Cascading Style Sheet (CSS)) see annex 3 for the source codes.

In realization of landuse planning model all inputs area aggregated in one form, even if they are required for intermediate tasks. This is because web entry which was created to allow external users to provide inputs for initiating the workflow from an external web page is only used to start the case.
Once the submit button id clicked the API is triggered to check if delegate activity in workflow is representing geo-process service by calling a function called *CheckTaskType()*. Listing 1 shows the *CheckTaskType()* function but the function call can be found in annex 2 of this research.

Listing 1: CheckTaskType() function.

```
function CheckTaskType()
{
    $conn = mysql_connect('127.0.0.1:3309','root', 'Aphro@rnra2013') or
            die('Could not connect: '. mysql_error()); // Create a connection to workflow model database
    mysql_select_db('wf_workflow');//select workflow model database when connection is established

    $result = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by APP_NUMBER desc limit 1") or
              die("Error: Unable to query the USER table.\n");

    $record = mysql_fetch_array($result, MYSQL_ASSOC);
    $CaseID= $record['APP_UID']; // Current CasID returned by process engine
    // query the delagated index for running task in particular caseID
    $result2 = mysql_query("SELECT DEL_INDEX
    FROM app_delegation
    WHERE APP_UID ='$CaseID'
    ORDER BY DEL_INDEX DESC
    LIMIT 1") or
              die("Error: Unable to query the app_delegation table.\n");//Error control
    $record2 = mysql_fetch_array($result2, MYSQL_ASSOC);
    $IndexID= $record2['DEL_INDEX'];
    // quert the running TaskID (this is the task with top delagation index)
    $result3 = mysql_query("SELECT TAS_UID FROM  wf_workflow.app_delegation where APP_UID='$CaseID' and DEL_INDEX=$IndexID") or
                die("Error: Unable to query  app_delegation table.\n");
    $record3 = mysql_fetch_array($result3, MYSQL_ASSOC);
    $TaskID=$record3['TAS_UID'];
    //Query Task type of the running task in land use planning model
    $result4 = mysql_query("SELECT ACT_TASK_TYPE FROM  wf_workflow.bpmn_activity where ACT_UID='$TaskID'") or
                die("Error: Unable to query  bpmn_activity table.\n");
    $record4 = mysql_fetch_array($result4, MYSQL_ASSOC);
    $TaskType=$record4['ACT_TASK_TYPE'];
    // query which task name associated with such particular running task

    return $TaskType;
}
```

After knowing the type of task, the API function *InstantiateTask()* is called to determine if the BPMN task is representing geo-service activity in workflow and if so extract the required data from its variables by calling *getVariables()*. Listing 2 show the *InstantiateTask()* function and the function call can be found in annex 3.

Listing 2: InstantiateTask() function

```
function InstantiateTask()
{
    $TaskType=CheckTaskType();
     $conn = mysql_connect('127.0.0.1:3309','root', 'Aphro@rnra2013') or
             die('Could not connect: '. mysql_error()); // Create a connection to workflow model database
     mysql_select_db('wf_workflow');//select workflow model database when connection is established
    // query which task name associated with such particular running task
    $result = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by APP_NUMBER desc limit 1") or
                  die("Error: Unable to query the USER table.\n");
     $record = mysql_fetch_array($result, MYSQL_ASSOC);
     $CaseID= $record['APP_UID']; // Current CasID returned by process engine
     // query the delagated index for running task in particular caseID
    $result2 = mysql_query("SELECT DEL_INDEX
    FROM app_delegation
    WHERE APP_UID ='$CaseID'
    ORDER BY DEL_INDEX DESC
    LIMIT 1") or
                  die("Error: Unable to query the app_delegation table.\n");//Error control
     $record2 = mysql_fetch_array($result2, MYSQL_ASSOC);
     $IndexID= $record2['DEL_INDEX'];
     // quert the running TaskID (this is the task with top delagation index)
    $result3 = mysql_query("SELECT TAS_UID FROM  wf_workflow.app_delegation where APP_UID='$CaseID' and DEL_INDEX=$IndexID") or
                  die("Error: Unable to query  app_delegation table.\n");
    $record3 = mysql_fetch_array($result3, MYSQL_ASSOC);
    $TaskID=$record3['TAS_UID'];
    if ($TaskType=="SERVICETASK")
    {
    $result5 = mysql_query("SELECT ACT_NAME FROM  wf_workflow.bpmn_activity where ACT_UID='$TaskID' and ACT_NAME LIKE 'WPS%'") or
                  die("Error: Unable to query the bpmn_activity table.\n");
    $record5 = mysql_fetch_array($result5, MYSQL_ASSOC);
    $ActivityName=$record5['ACT_NAME'];
    }
    return $ActivityName;
}
```

API *getVariables()* function extract the required variables of BPMN task representing geo-service and save it to global variable. Here is the simplified code of *getVariables()* function, moreover, the function call can be found in annex 3.

Listing 3: GetVariables() Function

```
function GetVariables($params2)
{
require_once('credentials.php');
$client = new SoapClient('http://130.89.229.224:8084/sysworkflow/en/green/services/wsdl2');
$params = array(array('userid'=>$username, 'password'=>$password));
$result = $client->__SoapCall('login', $params);
if ($result->status_code == 0)
    $sessionId = $result->message;
 else
    print "Unable to connect to ProcessMaker.\nError Number: $result->status_code\n" . "Error Message: $result->message\n";
 $params = array(array('sessionId'=>$sessionId));
 $result = $client->__SoapCall('caseList', $params);
 $casesArray = $result->cases;
 $conn = mysql_connect('130.89.229.224:3309','root', 'Aphro@rnra2013') or die('Could not connect: '. mysql_error());
     mysql_select_db('wf_workflow');
     $result = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by APP_NUMBER desc limit 1") or
                 die("Error: Unable to query the USER table.\n");
 $record = mysql_fetch_array($result, MYSQL_ASSOC);
 $CaseID= $record['APP_UID']; // Current CasID from processmaker database
   if ($casesArray != (object) NULL)
   {
       foreach ($casesArray as $case)
         if ($case->guid==$CaseID)
            $index=$case->delIndex ;
            $caseId=$case->guid;
    }
class variableStruct {
  public $name;
}
$vars = $params2;
                    $variables = array();
                     foreach ($vars as $var)
                     {
                        $obj = new variableStruct();
                        $obj->name = $var;
                        $variables[] = $obj;
                     }
 $params = array(array('sessionId'=>$sessionId, 'caseId'=>$caseId,
    'variables'=>$variables));
 $result = $client->__SoapCall('getVariables', $params);
 if ($result->status_code == 0 && $result->variables != (object) NULL)
 {
     $variablesArray = $result->variables;
 }
 return $variablesArray;
 }
```

Once the require process data are available API *ExecuteService()* function is called to trigger the execution of python script which sends WPS post request to WPS server (see Listing 4 for the simplified code of this function)

Listing 4: simplified codes for ExecuteService()

```
function ExecuteService($Parameters)
{
  $param1=$Parameters[0];
  $param2=$Parameters[1];
  $param3=$Parameters[2];
    $command="python  WPSBuffer.py $param1 $param2 $param3";
    $Status='';
    ob_start();// prevent outputting till you are done
    passthru($command);
    // get the result out
    $Status=ob_get_contents();
    ob_end_clean();// clean up the Status
     if ($Status==200)
                                {
                                    $resultUrl="http://win371.ad.utwente.nl/student/s6019978/Thesis/BoundaryBufferresult.xml";
                                    print("Task executed sucessfully!!!");
                                   $params = array(array('sessionId'=>$sessionId,'caseId'=>$CaseID, 'delIndex'=>$index));
                                   $result = $client->__SoapCall('routeCase', $params);
                                   if ($result->status_code == 0)
                                    {
                                        print '<br/>';
                                        print "Case derived: $result->message \n";
                                        echo '<form action="http://win371.ad.utwente.nl/student/s6019978/Thesis/APIControlScript.php"><input type="submit"
                                        value="Next Task" /></form>';
                                    }
                                        else
                                        print "Error deriving case: $result->message \n";
                                }
                                else
                                {
                                echo "Sorry Unable tocall python Script";
                                }

return $Status;
```

The call of this script triggers the execution of python script which sends WPS post request to Geo-Server which was used as WPS server in this thesis. This script takes input parameters from API *ExecuteService()* and API *getVariables()* function depending on the WPS operation to execute. See the codes for WPSBuffer python script but function call can be found in annex 3.

Listing 5: WPSBuffer python script

```
import requests
import sys
import urllib.request
import cgitb; cgitb.enable()
import  cgi
import json
import xml.etree.ElementTree as ET
try:
    tree = ET.ElementTree(file='InputTest.xml')
    WFS_AdminBoundary_Url=sys.argv[1].replace(",","&") # Get WFS getFeature URL from land use planning model
    DistanceBuffer=float(sys.argv[2]) #Get Distance variable
    Base_url=sys.argv[3]
    form = cgi.FieldStorage()
    root = tree.getroot()
    ######################################################################
    for child in root.iter():
        #print(child.tag,child.attrib,child.text)
        if ((child.tag=='{http://www.opengis.net/wps/1.0.0}LiteralData')and (child.text is None)):
            child.text=str(DistanceBuffer)
        if child.tag=='{http://www.opengis.net/wps/1.0.0}Reference':
            textvalue=child.attrib
            if not textvalue['{http://www.w3.org/1999/xlink}href']:
                textvalue['{http://www.w3.org/1999/xlink}href']=WFS_AdminBoundary_Url
    tree.write('output.xml')
    file=open('output.xml','r')
    k=file.read() hhhhhhhhhhh
    headers={'Content-Type': 'application/xml'}
    # Send WPS execute operation to the WPS Server
    resp=requests.post(Base_url,data=k,headers=headers)
    print(resp.status_code)
    #Save result in user created Folder
    file=open('BoundaryBufferresult.xml','w')
    file.write(resp.text)
    file.close()
except Exception as e:
```

In fact, API was created to allow remote access to BPMN element, services and identify the BPMN task representing geo-service in the process workflow. Moreover, it also makes service call of identified geo-service in workflow through a python script. In addition, python Script was developed to send WPS execute request to WPS server. Below you can found a figure 5.6 showing how WPS operations are executed sequentially based on their order in the organizational workflow.

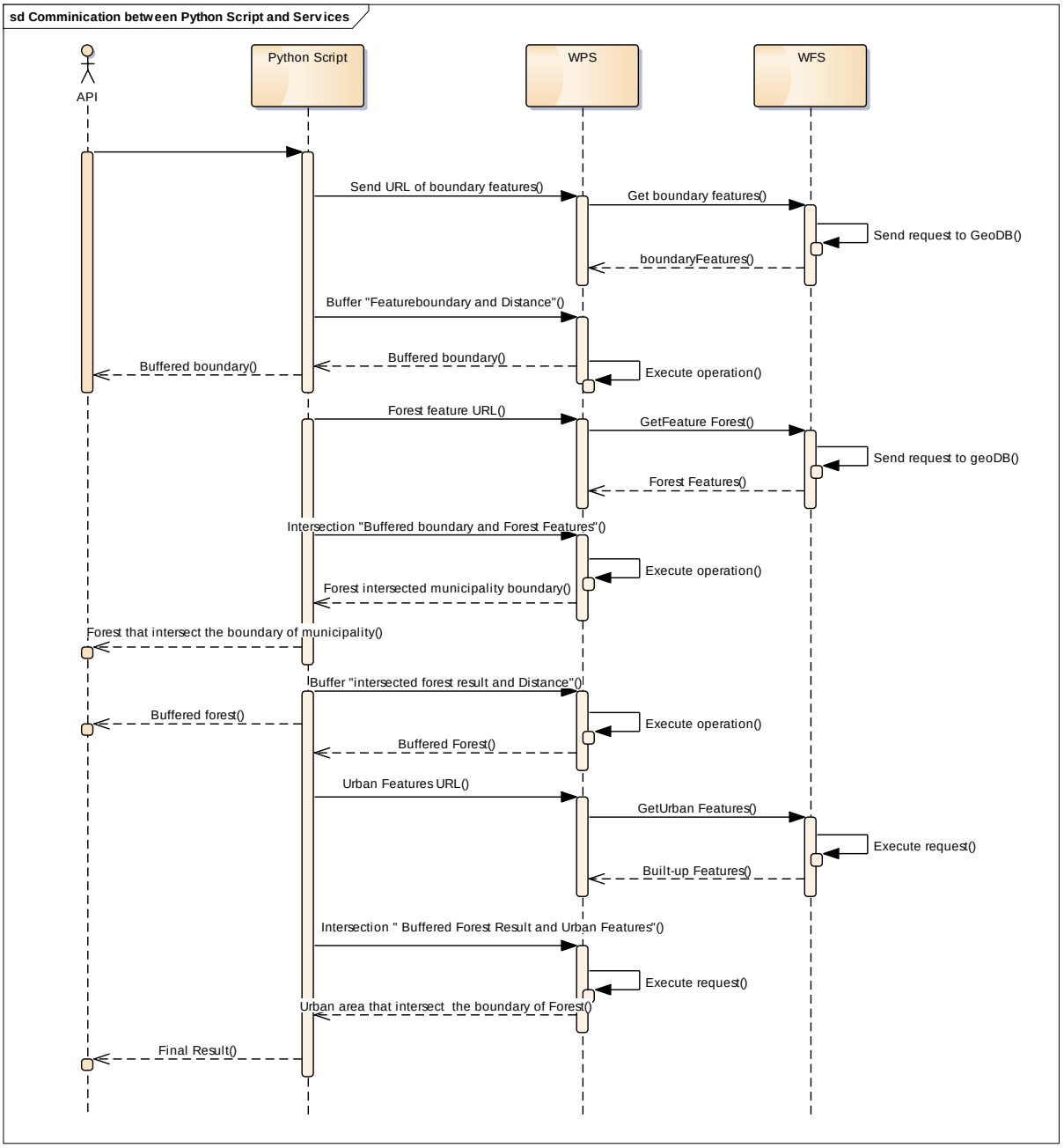


Figure 5.6: Sequence diagram showing the execution order of WPS services

WPS execute post request sends three input parameters to WPS Server: address of WPS server, data and content-type. Data is an XML files which defines the operation to be performed, the required input dataset or any other inputs, Data types of inputs and the format of the result as we discussed in section 2.4. See below XML file send to WPS server.

Listing 6: XML file sent to WPS Server as data in post request

```
<?xml version="1.0" encoding="UTF-8"?>
<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.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc" xmlns:wcs="http://www.opengis.net/wcs/1.1.1"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:BufferFeatureCollection</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="application/wfs-collection-1.0" xlink:href="WFS Feature" method="GET"/>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Buffer Distance</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput  mimeType="application/json">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

If the script is successfully executed API hands the execution back to process engine to continue executing the rest of the workflow by executing the code below. If an error occurred during execution this function will send an error message to the planner.

Listing 7: codes showing handover of execution of the workflow between API and process engine

```
$conn = mysql_connect('127.0.0.1:3309','root', 'Aphro@rnra2013') or  die('Could not connect: '. mysql_error());
     mysql_select_db('wf_workflow');
     $result = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by APP_NUMBER desc limit 1") or
               die("Error: Unable to query the USER table.\n");
        $record = mysql_fetch_array($result, MYSQL_ASSOC);
        $CaseID= $record['APP_UID'];
function NextTask($CaseID)
{
$Scriptresult=ExecuteService();
if($Scriptresult==200)
{
require_once('credentials.php');
$client = new SoapClient('http://127.0.0.1:8084/sysworkflow/en/green/services/wsdl2');
$params = array(array('userid'=>$username, 'password'=>$password));
$result = $client->__SoapCall('login', $params);
if ($result->status_code == 0)
    $sessionId = $result->message;
 else
    print "Unable to connect to ProcessMaker.\nError Number: $result->status_code\n" ."Error Message: $result->message\n";
     #Route case
     $params = array(array('sessionId'=>$sessionId,
         'caseId'=>$CaseID, 'delIndex'=> $index));
     $result = $client->__SoapCall('routeCase', $params);
     if ($result->status_code == 0)
     {
      print "Case derived: $result->message \n";
     }
  else
      print "Error deriving case: $result->message \n";
}
else
 echo "Sorry Unable tocall python Script";
}
```

The result or output of each processing unit is saved in created folder and file on the server and the URL link of the result is sent to process model variable repository by call API SendResult() function and it can be used as input to the next processing unit or activities as illustrated in Figure 5.1 and 5.2. These figures show the sequence and the execution order of different tasks in the process model. Below are codes for sending execution result URL model variable repository.

Listing 8: Send Result to Process repository

```
function SendResult($input,$input2)
{
 $ScriptStatus=ExecuteService();
 if ($ScriptStatus==200)
 {
 class variableListStruct {
     public $name;
     public $value;
 }

 $BufferResult = new variableListStruct();
 $BufferResult->name = "$input";
 $BufferResult->value = "$input2";
 $variables = array($BufferResult);
 $params = array(array('sessionId'=>$sessionId, 'caseId'=>$caseId,'variables'=>$variables));

 $result = $client->__SoapCall('sendVariables', $params);
 if ($result->status_code != 0)
     print "Error: $result->message \n";
}
else
{
        echo "Sorry Unable tocall python Script";
}
}
```

In fact, when planner sent inputs parameters to land use planning workflow the submit button triggers the execution of the workflow. Activities in the workflow are executed sequentially based on their sequence see figure 5.2, for landuse planning workflow execution order. As longer as the workflow is executed and some tasks are completed, if needed the intermediate status of the workflow can be visualized by the planner and it will show the status using a different colour (see figure 5.7 and 5.8).

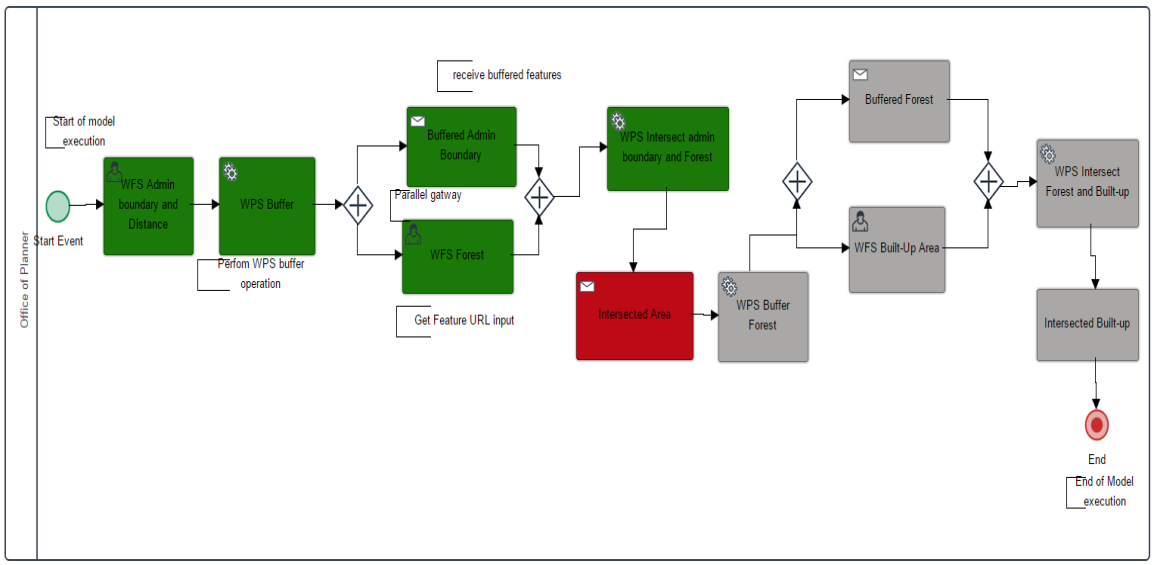


Figure 5.7: The status of task in BPMN diagram during the model execution

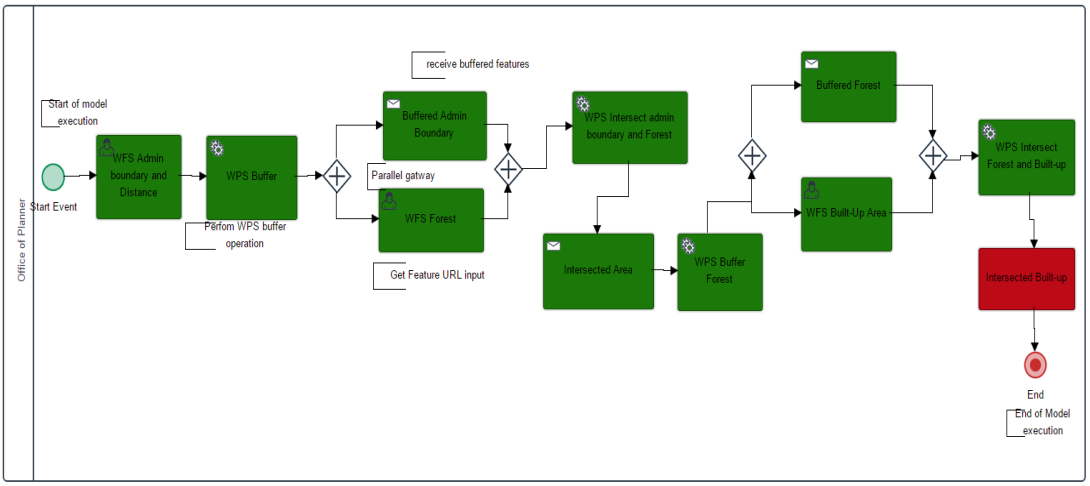
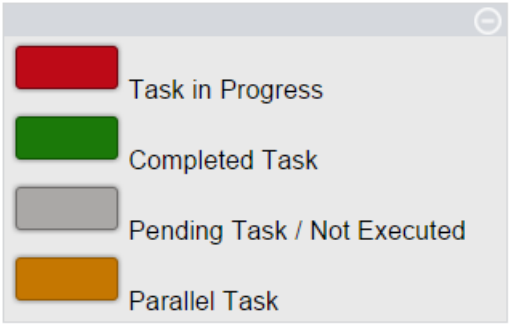


Figure 5.8: The status of task in BPMN diagram at the end model execution

Legend



The explained processes will be repeated as long as the workflow is running. After the final step of land use planning model execution, all results of WPS process units in the workflow are saved in workflow variables repository. The final result returned by land use planning workflow execution is in GML format, and it can be downloaded or visualized in web form as shown in figure 5.11. Different interfaces of each operation in the workflow are also captured as it is shown in figure 5.9 and 5.10. Moreover, as we discussed before the purpose of this scenario was to demonstrate if the proposed method for including geoprocessing services (WPS) in BPMN works. Therefore, as a result, the planner is able to visualize part of the urban area that is affected by the expansion of forest area in a new implementation of Landuse plan under construction in the municipality of Enschede. In addition, the planner also is able to see different interface showing the result of each service execution in the workflow as shown in figure 5.9 and 5.10 and the codes are provided in annex 3 of this thesis.

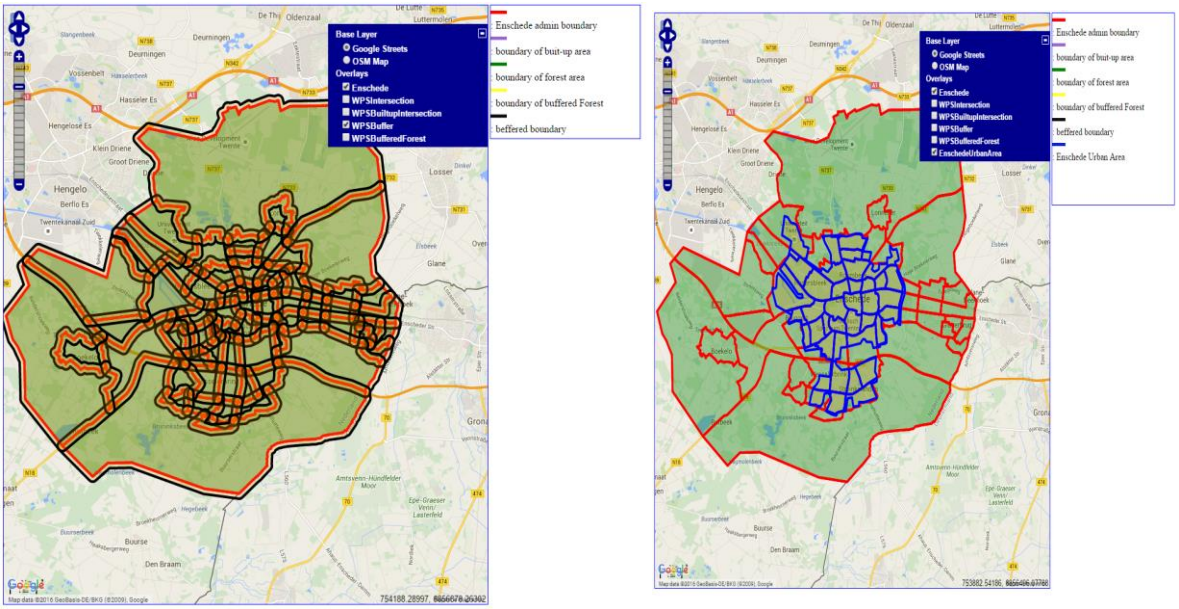


Figure 5.9: Buffered Enschede boundary and Enschede urban area

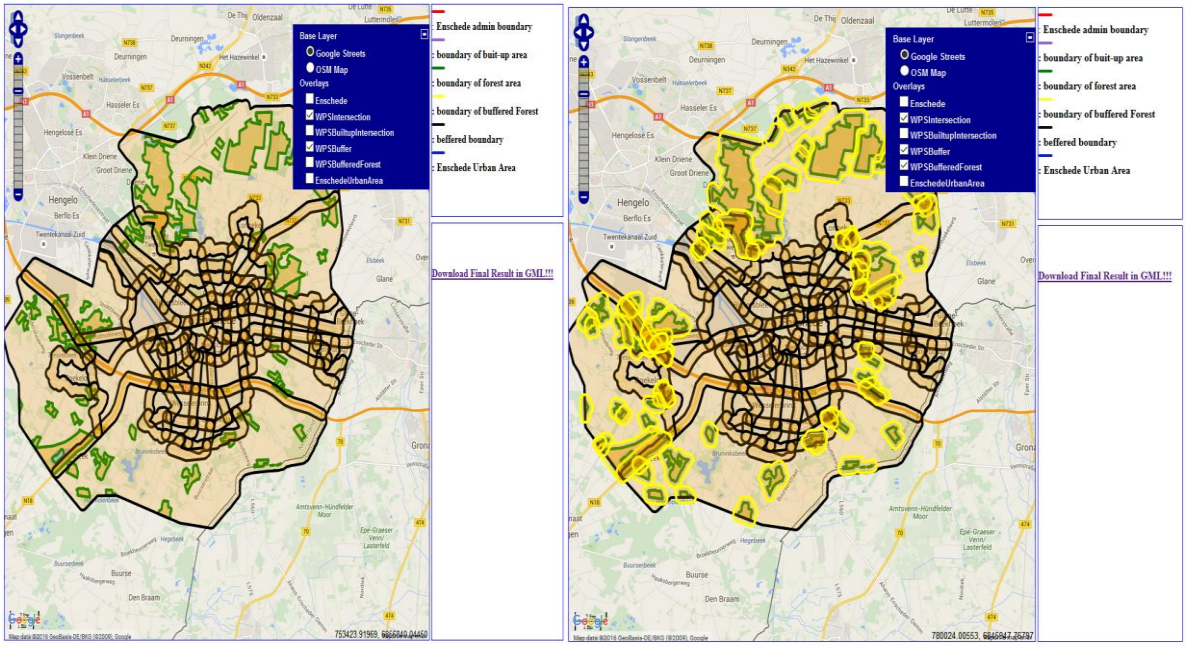
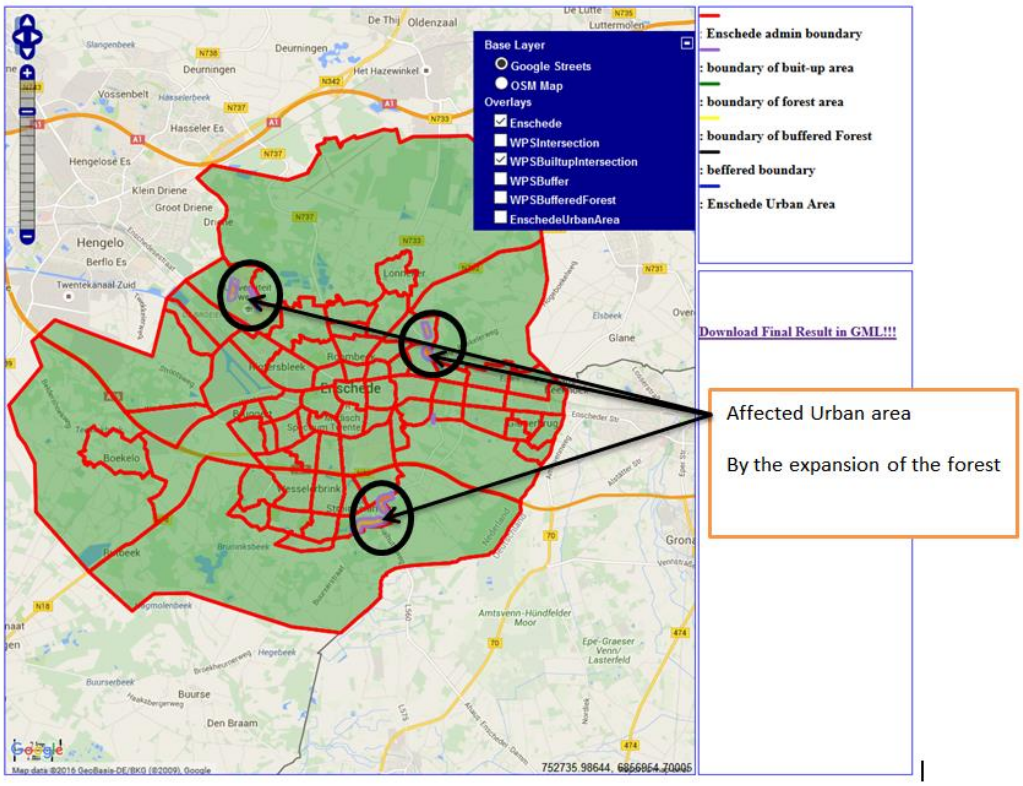


Figure 5.10: Forest that intersects the boundary of Enschede and buffered Forest



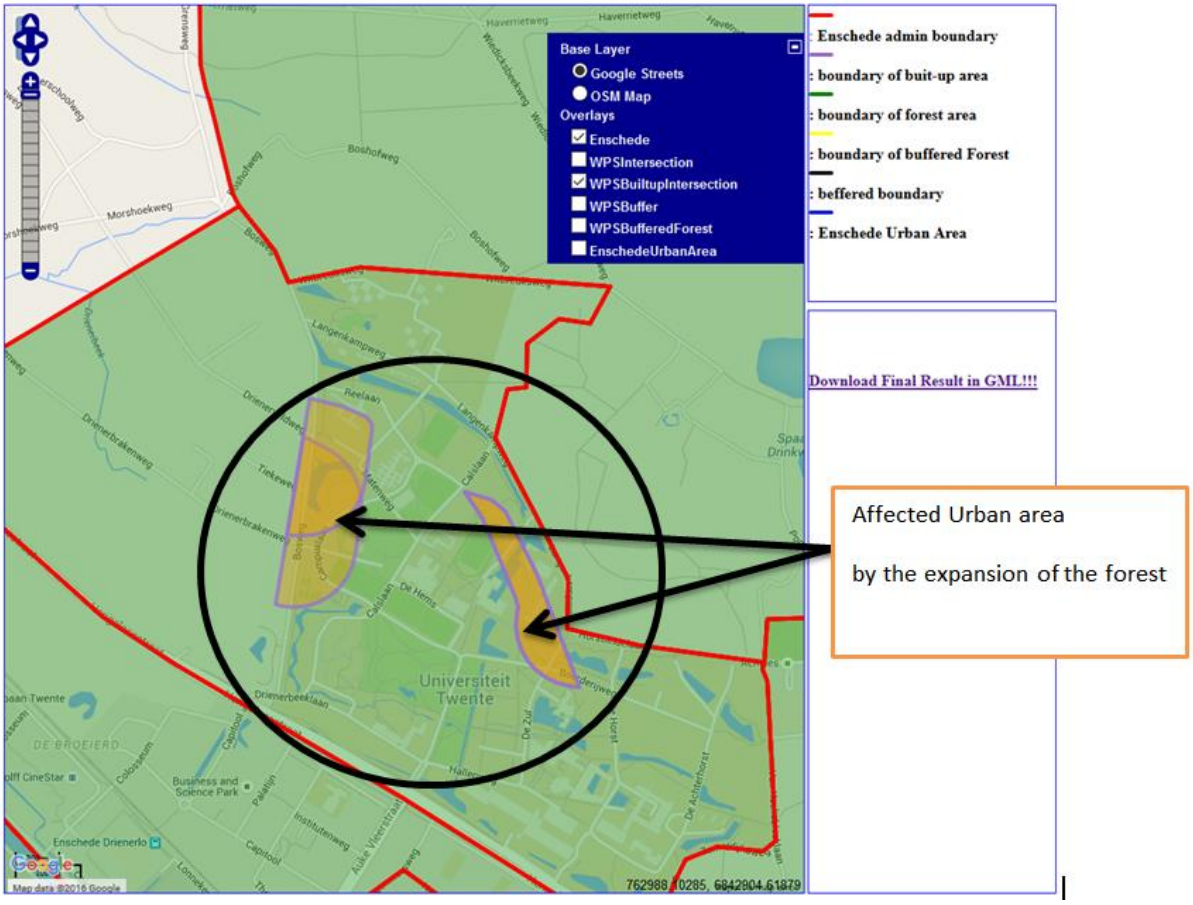


Figure 5.11: Final Map showing part of urban area affected by expansion of forest

5.3 Summary

In this chapter, we implemented and tested with an example as proof-of-concept a new proposed method in this research for handling WPS services with BPMN. An API with different functions was developed to identify BPMN tasks representing geoprocessing services activities in workflow and make the call of services through a developed python script. Land use planning workflow was modelled with BPMN notation and all geo-services within the workflow was identified and executed flowing the approach developed in this research. To that end, the planner is able to visualize the result of workflow execution on a web page or downloading a GML file corresponding to the final result. The codes used to implement both API, Landuse planning, a python script and result visualization codes can be found in the annex 1,2,3,4 of this thesis.

6 DISCUSSION, CONCLUSION, AND RECOMMENDATION

6.1 Introduction

In this research, we have proposed a method for extending the BPMN notation to include modelling elements for geoprocessing services. This way BPMN can be used to model all elements of a process including inputs, outputs and activities at the information viewpoint for geospatial workflows. Once BPMN model is completed, it can be orchestrated. During the orchestration, tasks that correspond to geoservices are identified to instantiate their corresponding service calls. The communication between BPMN model and the geoservices is realized through the API that accompanies our method.

6.2 Discussion

To realize the research objective, the flowing questions were addressed:

1. **What are the requirements to create a service chain?**

When we started this research, it was not possible to orchestrate services chain containing OGC services using BPMN. We looked into the existing functionality of BPMN, and into the specifications of OGC processing services, to understand why they cannot work together. We determined that to create OGC enabled service chains with BPMN, we need to include a BPMN element with new semantics so that it can be used to identify tasks involving OGC services in a workflow and link them to their corresponding OGC interface to make the necessary service calls. In additional to the new BPMN element's semantics, a set of functions were identified that were required to realize the communication with the actual services. All other aspects of the interaction between OGC services, such as their execution order or sequential constraints can be handled with standard BPMN elements.

2. **How to use standardized and readily understandable graphical notation like (BPMN), to represent geo-processes and their workflow?**

When an organization need to run workflows using BPMN and wanted to include geoprocessing services in those workflows this organization will have troubles. To execute BPMN tasks in a chain that correspond to OGC services, an application programming interface (API) is needed. This application is triggered when the workflow is being orchestrated and controls the communication between workflow and the remote services. This application also extracts the information from the tasks in the workflow and construct the appropriate request to instantiate the services. At the same time, it is responsible for inserting back into the workflow the response of the services.
This communication is done in two steps:

- API extracts the data from the workflow and,
- Python script writes services requests and redirects the responses.

3. What are the procedures of chaining geo-services with BPMN?

To chain a set of geoprocessing unit into a service chain, it is necessary to model the functional properties which enable two or more activities to interact each other within the chain. Here we listed the main procedures:

Process Mapping

Process mapping is the first activities within BPMN to model processes, at this level we:

- Identified all BPMN elements required to model processes based on organizational workflow.
- We created a workflow using the BPMN editor, for example, ProcessMaker. The result of this step is BPMN diagram showing the sequence of different services.

Process Description

Given that, all processing units services and their inputs was modelled using BPMN notation, at this level process is extended with enough information, such as marking the tasks representing geo-services, create variables to contain inputs and output from processing units execution, create dynamic form, step, users process rules, and assign tasks to corresponding users etc.

Process Model

This is what in our case we call workflow of activity or BPMN diagram, at this level the workflow, is equipped with enough information which makes it ready for being analysed and executed. BPMN has essential element to execute the workflow, but is not able to distinguish, activities representing geo-services to make service call as it was not designed for that. Therefore, the API we developed is involved to identify all tasks representing geo-service activity in workflow and read their properties to make appropriate service call. This element is equipped with enough function to allow connection of BPMN tasks representing geoservices in BPMN diagram (workflow) to their corresponding services. In addition to that all involved geo-services were described based on OGC standard and service requester is defined to be able to find out about the process status and, in case the process has completed, where to pick the result. Finally, a python script is trigged to send service execution request to the server.

4. How can web service orchestration improve the exploitation of OGC services?

Web service orchestration is the key for assembling geo-services into useful geoprocessing workflows. According to OGC WPS can be incorporated into service chains to make a sequence of web services. In service chaining, a set of services needs to be combined to work together to do the job. When this composition is done manually it becomes very complex and error prone. Therefore, we created the mechanism of putting those services together to avoid errors and make workflow easy to execute. To do that we basically focused on how BPMN and OGC services can understand each other to create a workflow containing spatial data and geoprocessing services based on organizational workflow. To create such sequence of services requires the analyses of what OGC provides and how a service can be requested based on the standard.

5. How to build and orchestrate the service chain into an executable workflow?

The main solution adopted for executing BPMN models up to version 1.2 was through their mapping to another language but with BPMN 2.0 is possible to obtain semi-automatically XML codes to deploy in orchestration engine. BPEL and XPDL are aimed to describe the executable workflows the specification of low-level implementation details such as calling a web services or geo-services, gathering their responses and handling the proper transformation operations.

In this research, as BPMN was not natively created to model geo-services different way was proposed and implemented. To this end, API was developed to identify BPMN tasks representing geo-service in workflow, describe the individual components by providing all details and instantiate these geo-services to make service call through a python script.
At the same time, it is responsible for inserting back into the workflow the response of the services. ProcessMaker engine runs a workflow and returns an index of delegated task to the API through ProcessMaker web service. In the end, the result from execution is saved and web interface was developed to help the user to visualize the result from the execution.

6.3 Conclusion

We have studied the functionalities and the limitations of BPMN to model geoprocesses. To that end, those limitations were removed by new proposed method in section 4.3 of this thesis. In that section, we have proposed and develop an API accompanies with different functions able to identify BPMN element representing geoservices to make a service call. In the same way, a python script was developed to send the execute request to WPS server. A use case was implemented in chapter 5 to demonstrate the functionality of our developed method. Moreover, it has been tested with a single use case. Based on that we concluded that the limitation of BPMN to model geoprocessing services was removed now, it is no longer a problem to model WPS services using BPMN.

6.4 Recommendation

In order to improve the outcome of this research project, we recommend the following:

Implementation of "geo-task"

For none technical persons (persons with no programming background) to be able to benefit both standard BPMN and OGC services, A "geo-task" need to be developed and added in BPMN as a spatial task or a task that has spatial information. A "geo-task" in this case is a type of task with clear semantics to recognize, describe the spatial data and spatial services in the context of the business process workflow. Then, during the execution of the workflow, the specialized BPMN element to run workflow can recognize it as spatial task or task that has spatial information such as Location etc.

Extending BPMN business rule task to support geospatial data and services

BPMN provides a business rule task that defines a mechanism for a process to provide inputs to a business rule engine and get the output after of the computation. The interested MSc student can study the functionality of this task and adds more functionality to make it useful for supporting spatial data and spatial services.

Extend the functionality of new proposed method

Even if the proposed method is good enough to chain WPS services using BPMN, the current functionality of the new proposed method can be enhanced by adding more functionality or components where necessary.
For example, python script developed to send WPS execute request need to be enhanced to describe all possible number of WPS operations, marking service task representing geo-services in BPMN diagram need to be enhanced by providing a significant symbol to differentiate service task representing geo-service and normal service task. In addition, to that error handling and input validation mechanism need to be improved and, if possible, include non-functional requirements such as security and quality of service etc.

LIST OF REFERENCES

Abdaldhem Albreshne, Patrik Fuhrer, J. P. (2009). Web Services Technologies: State of the Art. Retrieved January 28, 2016, from http://diuf.unifr.ch/drupal/softeng/sites/diuf.unifr.ch.drupal.softeng/files/file/publications/internal/WP09-04.pdf

Alameh, N. (2003). Chaining geographic information web services. *IEEE Internet Computing*, *7*(5), 22–29. doi:10.1109/MIC.2003.1232514

Albrecht, J., Derman, B., & Ramasubramanian, L. (2008). Geo-ontology tools: The missing link. doi:10.1111/j.1467-9671.2008.01108.x

Allweyer, T. (2010). *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. BoD – Books on Demand. Retrieved from https://books.google.com/books?hl=en&lr=&id=fdlC7K_3dzEC&pgis=1

Barker, A., & Van Hemert, J. I. (2007). Scientific Workflow: A Survey and Research Directions. doi:10.1007/978-3-540-68111-3

Baumann, P. (2012). OGC® WCS 2.0 Interface Standard- Core: Corrigendum. Retrieved January 31, 2016, from http://www.opengeospatial.org/standards/wcs\npapers2://publication/uuid/D303A640-AF41-432D-B72C-593E1BDCFF47

Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., & Orchard, D. (2004). Web Services Architecture. Retrieved October 16, 2015, from http://www.w3.org/TR/ws-arch/#whatis

Campagna, M. (2014). Orchestrating the spatial planning process : from Business Process Management to 2 nd generation Planning Support Systems. Retrieved August 18, 2015, from http://www.agile-online.org/Conference_Paper/cds/agile_2014/agile2014_132.pdf

Campagna, M., Ivanov, K., & Massa, P. (2014). Implementing Metaplanning with Business Process Management. *Procedia Environmental Sciences*, *22*, 199–209. doi:10.1016/j.proenv.2014.11.020

Chinosi, M., & Trombetta, A. (2012). BPMN: An introduction to the standard. doi:10.1016/j.csi.2011.06.002

Coalition, W. M. (2012). Workflow Standard Process Definition Interface--XML Process Definition Language. Retrieved October 26, 2015, from http://www.xpdl.org/nugen/p/gseonklyf/a/2009-4-7 xpdl 2.doc

Database and Expert Systems Applications: 17th International Conference, DEXA 2006, Krakow, Poland, September 4-8, 2006, Proceedings. (2006). Springer Science & Business Media. Retrieved from https://books.google.com/books?id=W07Q1D18Z80C&pgis=1

Date, S., Date, A., Date, P., Editor, C. S., Liang, S., Huang, C., & Khalafbeigi, T. (2015). Open Geospatial Consortium. Retrieved January 31, 2016, from http://docs.opengeospatial.org/is/09-025r2/09-025r2.html

de By, R. a., Lemmens, R., & Morales, J. (2009). A skeleton design theory for spatial data infrastructure: Methodical construction of SDI nodes and SDI networks. *Earth Science Informatics*, *2*(4), 299–313. doi:10.1007/s12145-009-0034-7

Díaz, L., Pepe, M., Granell, C., Carrara, P., & Rampini, A. (2010). Developing and chaining web processing services for hydrological models. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, *38*(4W13). Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84923870654&partnerID=tZOtx3y1

Donaubauer, A., & Straub, F. (2010). A Spatial Decision Service for BPEL. Retrieved October 17, 2015, from http://www.isprs.org/proceedings/xxxviii/4-W13/ID_40.pdf

Foerster, T., Schaeffer, B., Brauner, J., & Jirka, S. (2009). Integrating OGC web processing services into geospatial Mass-market applications. *Proceedings of the International Conference on Advanced Geographic Information Systems and Web Services, GEOWS 2009*, 98–103. doi:10.1109/GEOWS.2009.19

Friis-Christensen, A., Ostländer, N., Lutz, M., & Bernard, L. (2007). Designing service architectures for distributed geoprocessing: Challenges and future directions. *Transactions in GIS*, *11*(6), 799–818. doi:10.1111/j.1467-9671.2007.01075.x

Georgakopoulos, D., Ritter, N., & Benatallah, B. (2007). *Service-Oriented Computing ICSOC 2006: 4th International Conference, Chicago, IL, USA, December 4-7, 2006, Workshop Proceedings*. Springer Science & Business Media. Retrieved from https://books.google.com/books?id=rc3Y9Mthsc8C&pgis=1

GIS HYDRO 2009. (2009). Retrieved December 21, 2015, from https://www.crwr.utexas.edu/gis/gishydro09/what_is_SOA.html

Hallwyl, T., Henglein, F., & Hildebrandt, T. (2010). A standard-driven implementaion of WS-BPEL 2.0. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10* (p. 2472). New York, New York, USA: ACM Press. doi:10.1145/1774088.1774599

Hu, L., Yue, P., & Gong, J. (2013). Asynchronous geoprocessing services: An interoperable approach. In *2013 Second International Conference on Agro-Geoinformatics (Agro-Geoinformatics)* (pp. 408–412). IEEE. doi:10.1109/Argo-Geoinformatics.2013.6621953

International Organization for Standardization. (2005). ISO 19119:2005 Geographic information -- Services. Retrieved January 4, 2016, from http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39890

Jordan, D., & Alves, A. (2007a). Web Services Business Process Execution Language Version 2. 0. doi:10.1146/annurev.biophys.37.032807.125832

Jordan, D., & Alves, A. (2007b). Web Services Business Process Execution Language Version 2. 0. doi:10.1146/annurev.biophys.37.032807.125832

Josuttis, N. M. (2007a). Soa in practice: the art of distributed system Design. doi:citeulike-article-id:2722436

Josuttis, N. M. (2007b). Soa in practice: the art of distributed system Design. doi:citeulike-article-id:2722436

Kagoyire, C. (2009). *Web geoprocessing services on GML with fast XML database*. University of Twente Faculty of Geo-information and Earth Observation (ITC). Retrieved from http://www.itc.nl/library/papers_2009/msc/gfm/kagoyire.pdf

Kudrass, T. (2003). Describing architectures using RM-ODP. Retrieved February 11, 2016, from http://www.imn.htwk-leipzig.de/~kudrass/Publikationen/OOPSLA99.pdf

Lemmens, R., de By, R., Gould, M., Wytzisk, A., Granell, C., & van Oosterom, P. (2007). Enhancing Geo-Service Chaining through Deep Service Descriptions. *Transactions in GIS*, *11*(6), 849–871. doi:10.1111/j.1467-9671.2007.01079.x

Lemmens, R., Wytzisk, A., By, R. d., Granell, C., Gould, M., & van Oosterom, P. (2006). Integrating Semantic and Syntactic Descriptions to Chain Geographic Services. *IEEE Internet Computing*, *10*(5), 42–52. doi:10.1109/MIC.2006.106

Meek, S., Jackson, M., & Leibovici, D. G. (2016). A BPMN solution for chaining OGC services to quality assure location-based crowdsourced data. *Computers & Geosciences*, *87*, 76–83. doi:10.1016/j.cageo.2015.12.003

Meng, X., Xie, Y., & Bian, F. (2010). Distributed geospatial analysis through web processing service: A case study of earthquake disaster assessment. doi:10.4304/jsw.5.6.671-679

Mukherjee, J., & Ghosh, S. K. (2010). Geospatial service chaining in decision support systems. In *Proceedings of the 2010 Annual IEEE India Conference: Green Energy, Computing and Communication, INDICON 2010* (pp. 1–4). IEEE. doi:10.1109/INDCON.2010.5712654

Mukherjee, J., Mukherjee, I., & Ghosh, S. K. (2011). Framework for spatial query resolution for decision support using geospatial service chaining and fuzzy reasoning. doi:10.1007/978-3-642-19423-8_9

Object Management Group, Parida, R., & Mahapatra, S. (2011). Business Process Model and Notation (BPMN) Version 2.0. doi:10.1007/s11576-008-0096-z

OGC. (2010). OpenGIS ® Web Map Tile Service Implementation Standard. Retrieved January 31, 2016, from http://www.opengeospatial.org/standards/wmts

OGC, Mueller, M., & Pross, B. (2015). OGC WPS 2.0 Interface Standard. Retrieved August 2, 2015, from http://www.opengeospatial.org/pressroom/pressreleases/2241

Ouyang, C., Aalst, V. Der, Wil, M. P., & Arthur, H. M. (2009). From business process models to process-oriented software systems: The BPMN to BPEL way. doi:10.1016/S0190-9622(06)01179-0

Prager, M., Klímek, F., & Růžička, J. (2009). GeoWeb Services Orchestration Based on BPEL or BPMN. Retrieved from http://www.researchgate.net/publication/229010710_GeoWeb_Services_Orchestration_Based_on_BPEL_or_BPMN

Růžička, J. (2009). ISO 19115 for GeoWeb services orchestration. *Geoinformatics FCE CTU*, *3*, 51–66. doi:10.14311/gi.3.5

Schaeffer, B. (2009). OGC OWS-6 Geoprocessing Workflow Architecture Engineering Report. Retrieved October 26, 2015, from https://portal.opengeospatial.org/files/?artifact_id=34968

Senkler, K. (2007). Open Geospatial Consortium Inc. OpenGIS ® Catalogue Services Specification 2. 0 . 2 - ISO Metadata Application Profile. Retrieved January 31, 2016, from http://www.opengeospatial.org/standards/cat\nhttp://portal.opengeospatial.org/files/?artifact_id=20555

Shapiro, R., White, S. a, Bock, C., Palmer, N., zur Muehlen, M., Brambilla, M., & Gagné, D. (2012). *BPMN 2.0 Handbook Second Edition: Methods, Concepts, Case Studies and Standards in Business Process Modeling Notation (BPMN)*. Future Strategies Inc. Retrieved from https://books.google.com/books?id=9U3DO5PoTDQC&pgis=1

Specification, A. (2001). Category : OpenGIS ® Implementation Specification Status : Adopted Specification Web Map Service Implementation Specification. Retrieved January 31, 2016, from https://earthdata.nasa.gov/files/01-068r3_web_map_service_implementation_specification.pdf

Stollberg, B., & Zipf, A. (2007). OGC Web Processing Service Interface for Web Service Orchestration Aggregating Geo-processing Services in a Bomb Threat Scenario. doi:10.1007/978-3-540-76925-5_18

Strickland, A., Whittington, D., Taylor, P., & Wang, B. (2006). Analysis of BPEL and High-Level Web Service Orchestration: Bringing Benefits to the Problems of the Business. In *Database and Expert Systems Applications* (Vol. 4080, pp. 123–137). doi:10.1007/11827405_13

Teams, D. (2004). Rational Unified Process Best Practices for Software. doi:10.1.1.27.4399

Westerholt, R., & Resch, B. (2015). Asynchronous Geospatial Processing: An Event-Driven Push-Based Architecture for the OGC Web Processing Service. *Transactions in GIS*, *19*(3), 455–479. doi:10.1111/tgis.12104

Ying, Y., Qunyong, W., & Linjun, K. (2012). Integrate geo-spatial web processing services by workflow technology. doi:10.1007/978-3-642-27323-0_41

Yu, G. E., Zhao, P., Di, L., Chen, A., Deng, M., & Bai, Y. (2012). BPELPower-A BPEL execution engine for geospatial web services. *Computers and Geosciences*, *47*, 87–101. doi:10.1016/j.cageo.2011.11.029

Zhao, P., Di, L., & Yu, G. (2012). Building asynchronous geospatial processing workflows with web services. *Computers & Geosciences*, *39*, 34–41. doi:10.1016/j.cageo.2011.06.006

Zhao, P., Foerster, T., & Yue, P. (2012). The Geoprocessing Web. doi:10.1016/j.cageo.2012.04.021

ANNEX 1: API Function Definitions

```
<?php
global $TaskType1;

// Function to check type of task
function CheckTaskType()
{
   $connection = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or
                              die('Unable to connect: '. mysql_error()); // Create a connection to workflow
model database
          mysql_select_db('wf_workflow');//select workflow model database when connection is
established

          $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                        die("Error: can't connect to application table.\n");

          $record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
          $CaseID= $record['APP_UID']; // Current CasID returned by process engine
          // query the delagated index for running task in particular caseID
         $result2 = mysql_query("SELECT DEL_INDEX
         FROM app_delegation
         WHERE APP_UID ='$CaseID'
         ORDER BY DEL_INDEX DESC
         LIMIT 1") or
                                        die("Error: Unable to query the app_delegation table.\n");//Error
control
          $record2 = mysql_fetch_array($result2, MYSQL_ASSOC);
          $IndexID= $record2['DEL_INDEX'];
   // quert the running TaskID (this is the task with top delagation index)
         $result3 = mysql_query("SELECT TAS_UID FROM  wf_workflow.app_delegation where
APP_UID='$CaseID' and DEL_INDEX=$IndexID") or
                                          die("Error: Unable to query  app_delegation table.\n");

         $record3 = mysql_fetch_array($result3, MYSQL_ASSOC);
         $TaskID=$record3['TAS_UID'];
         //Query Task type of the running task in land use planning model
         $result4 = mysql_query("SELECT ACT_TASK_TYPE FROM  wf_workflow.bpmn_activity
where ACT_UID='$TaskID'") or
                                          die("Error: Unable to query  bpmn_activity table.\n");
         $record4 = mysql_fetch_array($result4, MYSQL_ASSOC);
         $TaskType=$record4['ACT_TASK_TYPE'];
         // query which task name associated with such particular running task

         return $TaskType;
}
//Function that instantiate task

function InstantiateTask()
{
  $TaskType=CheckTaskType();
   $connection = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or
```

```
                        die('Could not connect: '. mysql_error()); // Create a connection to workflow
model database
         mysql_select_db('wf_workflow');//select workflow model database when connection is
established
         // query which task name associated with such particular running task
         $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                 die("Error: Unable to query the USER table.\n");
          $record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
          $CaseID= $record['APP_UID']; // Current CasID returned by process engine
          // query the delagated index for running task in particular caseID
         $result2 = mysql_query("SELECT DEL_INDEX
         FROM app_delegation
         WHERE APP_UID ='$CaseID'
         ORDER BY DEL_INDEX DESC
         LIMIT 1") or
                                 die("Error: Unable to query the app_delegation table.\n");//Error
control
          $record2 = mysql_fetch_array($result2, MYSQL_ASSOC);
          $IndexID= $record2['DEL_INDEX'];
    // quert the running TaskID (this is the task with top delagation index)
         $result3 = mysql_query("SELECT TAS_UID FROM  wf_workflow.app_delegation where
APP_UID='$CaseID' and DEL_INDEX=$IndexID") or
                                 die("Error: Unable to query  app_delegation table.\n");

         $record3 = mysql_fetch_array($result3, MYSQL_ASSOC);
         $TaskID=$record3['TAS_UID'];
         if ($TaskType=="SERVICETASK")
         {
         $result5 = mysql_query("SELECT ACT_NAME FROM  wf_workflow.bpmn_activity where
ACT_UID='$TaskID' and ACT_NAME LIKE 'WPS%'") or
                                 die("Error: Unable to query the bpmn_activity table.\n");
         $record5 = mysql_fetch_array($result5, MYSQL_ASSOC);
         $ActivityName=$record5['ACT_NAME'];
         }
         else
         {
         $result5 = mysql_query("SELECT ACT_NAME FROM  wf_workflow.bpmn_activity where
ACT_UID='$TaskID'") or
                                 die("Error: Unable to query the bpmn_activity table.\n");
         $record5 = mysql_fetch_array($result5, MYSQL_ASSOC);
         $ActivityName=$record5['ACT_NAME'];
         }
         return $ActivityName;
}
//Function that retrieves task properties

function GetVariables($params2)
{
require_once('Authentication.php');
$user = new SoapClient('http://130.89.231.83:8084/sysworkflow/en/green/services/wsdl2');
$parameters = array(array('userid'=>$username, 'password'=>$password));
$outcome = $user->__SoapCall('login', $parameters);
if ($outcome->status_code == 0)
   $sessionId = $outcome->message;
```

```
else
   print "Unable to connect to ProcessMaker.\nError Number: $outcome->status_code\n" .
       "Error Message: $outcome->message\n";
$parameters = array(array('sessionId'=>$sessionId));
$outcome = $user->__SoapCall('caseList', $parameters);
$casesArray = $outcome->cases;
$conn = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or
                              die('Could not connect: '. mysql_error());
          mysql_select_db('wf_workflow');
          $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                        die("Error: Unable to query the USER table.\n");

$record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
$CaseID= $record['APP_UID']; // Current CasID from processmaker database
  if ($casesArray != (object) NULL)
  {
     foreach ($casesArray as $case)
            if ($case->guid==$CaseID)
                    $index=$case->delIndex ;
                           $caseId=$case->guid;
          }

class Structvariable {
  public $name;
}
$vars = $params2;
                                                $variables = array();
                                                 foreach ($vars as $var)
                                                 {
                                                          $obj = new Structvariable();
                                                          $obj->name = $var;
                                                          $variables[] = $obj;
                                                 }
$parameters = array(array('sessionId'=>$sessionId, 'caseId'=>$caseId,
    'variables'=>$variables));
$outcome = $user->__SoapCall('getVariables', $parameters);
if ($outcome->status_code == 0 && $outcome->variables != (object) NULL)
{
     $variablesArray = $outcome->variables;
}
return $variablesArray;
}

//Function that returns a delegated task index

function ReturnIndex()
{
require_once('Authentication.php');
$user = new SoapClient('http://130.89.231.83:8084/sysworkflow/en/green/services/wsdl2'); // Login
Processmaker to access its functionality
$parameter = array(array('userid'=>$username, 'password'=>$password));
$outcome = $user->__SoapCall('login', $parameter);
if ($outcome->status_code == 0)
   $sessionId = $outcome->message;
```

```
 else
   print "Failed to connect to ProcessMaker.\nError Number: $outcome->status_code\n" .
       "Error Message: $outcome->message\n";
                    $parameter = array(array('sessionId'=>$sessionId));
 $outcome = $user->__SoapCall('caseList', $parameter);
 $casesArray = $outcome->cases;
 $connection = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or
                              die('Unable to connect: '. mysql_error());
          mysql_select_db('wf_workflow');
          $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                      die("Error: Unable to query the application table.\n");
 $record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
 $CaseID= $record['APP_UID']; // Current CasID from processmaker database
  if ($casesArray != (object) NULL)
  {
     foreach ($casesArray as $case)
            if ($case->guid==$CaseID)
                    $index=$case->delIndex ;
                          $caseId=$case->guid;
          }
          return $index;
}

// Function that send URL result of the executed service to process maker model repository

function SendResult($param2)
{
require_once('Authentication.php');
$user = new SoapClient('http://130.89.231.83:8084/sysworkflow/en/green/services/wsdl2');
$parameter = array(array('userid'=>$username, 'password'=>$password));
$outcome = $user->__SoapCall('login', $parameter);
if ($outcome->status_code == 0)
   $sessionId = $outcome->message;
 else
   print "Unable to connect to ProcessMaker.\nError Number: $outcome->status_code\n" .
       "Error Message: $outcome->message\n";

 class variableListStruct {
    public $name;
    public $value;
 }
 $vars = $param2 ;
 $variables = array();
 foreach ($vars as $key => $val)
 {
   $obj = new variableListStruct();
   $obj->name = $key;
   $obj->value = $val;
   $variables[] = $obj;
 }
 $parameter = array(array('sessionId'=>$sessionId, 'caseId'=>$caseId, 'variables'=>$variables));
 $outcome = $user->__SoapCall('sendVariables', $parameter);
 if ($outcome->status_code != 0)
    print "Error: $outcome->message \n";
```

```
 else
 print $outcome->message;
}

// Function that calls python script to execute service request
function ExecuteService($Parameters)
{
  for($i = 0, $c = count($array); $i < $c; $i++)
      var_dump($array[$i]);
 $param1=$Parameters[0];
 $param2=$Parameters[1];
 $param3=$Parameters[2];
          $command="python  WPSBuffer.py $param1 $param2 $param3";
          $Status='';
          ob_start();// prevent outputting till you are done
          passthru($command);
          // get the result out
          $Status=ob_get_contents();
          ob_end_clean();// clean up the Status
           if ($Status==200)
                                                                {

$resultUrl="http://win371.ad.utwente.nl/student/s6019978/Thesis/BoundaryBufferresult.xml";
                                                                        print("'Task executed sucessfully!!!");

                                                                                echo '<form
action="http://win371.ad.utwente.nl/student/s6019978/Thesis/APIControlScript.php"><input
type="submit" value="Next Task" /></form>';
                                                                }
                                                                else
                                                                {
                                                                echo "Sorry Unable tocall python Script";

                                                                }

return $Status;
}

//Function that move the case to the next task during workflow execution

function NextTask()
{
$connection = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or  die('Could not connect:
'. mysql_error());
          mysql_select_db('wf_workflow');
          $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                    die("Error: Unable to query the application table.\n");
                    $record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
                    $CaseID= $record['APP_UID'];
$Scriptresult=ExecuteService();
if($Scriptresult==200)
{
require_once('Authentication.php');
$user = new SoapClient('http://130.89.231.83:8084/sysworkflow/en/green/services/wsdl2');
```

```
$parameters = array(array('userid'=>$username, 'password'=>$password));
$outcome = $user->__SoapCall('login', $parameters);
if ($outcome->status_code == 0)
   $sessionId = $outcome->message;
 else
   print "Unable to connect to ProcessMaker.\nError Number: $outcome->status_code\n" ."Error
Message: $outcome->message\n";
            #Route case
            $parameters = array(array('sessionId'=>$sessionId,
                        'caseId'=>$CaseID, 'delIndex'=> $index));
            $outcome = $user->__SoapCall('routeCase', $parameters);
            if ($outcome->status_code == 0)
            {
     print "Case derived: $outcome->message \n";
            }
 else
     print "Error deriving case: $outcome->message \n";
}
else
 echo "Sorry Unable to call python Script";
}

// Function that route case

Function routeCase()
{
require_once('Authentication.php');
$user = new SoapClient('http://130.89.231.83:8084/sysworkflow/en/green/services/wsdl2');
$parameters = array(array('userid'=>$username, 'password'=>$password));
$outcome = $user->__SoapCall('login', $parameters);
if ($outcome->status_code == 0)
   $sessionId = $outcome->message;
 else
   print "Unable to connect to ProcessMaker.\nError Number: $outcome->status_code\n" ."Error
Message: $outcome->message\n";
          $connection = mysql_connect('130.89.231.83:3309','root', 'Aphro@rnra2013') or  die('Could not
connect: '. mysql_error());
            mysql_select_db('wf_workflow');
            $queryoutcome = mysql_query("SELECT APP_UID FROM  wf_workflow.application order by
APP_NUMBER desc limit 1") or
                                        die("Error: Unable to query the application table.\n");
                    $record = mysql_fetch_array($queryoutcome, MYSQL_ASSOC);
                    $CaseID= $record['APP_UID']; // Current CaseID from processmaker database
                    $outcome = $user->__SoapCall('caseList', $parameters);
     $casesArray = $outcome->cases;
  if ($casesArray != (object) NULL)
  {
     foreach ($casesArray as $case)
             if ($case->guid==$CaseID)
                     $index=$case->delIndex ;
                           $caseId=$case->guid;
          }
$parameters = array(array('sessionId'=>$sessionId,'caseId'=>$CaseID, 'delIndex'=>$index));
                                        $outcome = $user->__SoapCall('routeCase', $parameters);
                                        if ($outcome->status_code == 0)
```

```
                                        {
                                                echo '<form
action="http://win371.ad.utwente.nl/student/s6019978/Thesis/APIControlScript.php"><input
type="submit" value="Next Task" /></form>';
                                        }
}
 ?>
```

ANNEX 2: Landuse Planning Implementation Codes

```
<?php
global $index;
global $caseId;
global $ActivityName;
global $taskid;
global $WFSBoundary;
global $WFSDistance;
global $BufferResult;
global $WFSForestUrl;
global $WPSBuffer_Result;
global  $WFSForest;
global  $WPSForestresult;
global  $Base_Url;
global  $taskType;
global  $StatusID;
include 'API.php';
$index=ReturnIndex();
print $index;
            switch($index)
    {
            case 1:
          $taskType=CheckTaskType();
            //Check the returned task;
                              if ($taskType=="USERTASK")
          {
                               $ActivityName=InstantiateTask();
                               if ($ActivityName=="WFS Forest")
                               {
                                routeCase();// move to the next task
                               }
                               if ($ActivityName=="WFS Built-Up Area")
                               {
                                routeCase();//move to the next task
                               }
                               if ($ActivityName=="WFS Admin boundary and Distance")
                               {
                                routeCase();//move to the next task
                               }
                              }
                              elseif ($taskType=="RECEIVETASK")
           {
                                $ActivityName=InstantiateTask();//check the name of activity
                                      if ($ActivityName=="Buffered Admin Boundary")
                                      {

SendResult(array('WPSBuffer_Result'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/BoundaryBufferresult.xml'));
                                        routeCase();
                                      }
                                      if ($ActivityName=="Intersected Area")
                                      {
```

```
SendResult(array('WPSIntersectionResult'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/For
est_boundaryIntersection.xml'));
                                        routeCase();
                                       }
                                       if ($ActivityName=="Buffered Forest")
                                       {

SendResult(array('WPSForestBufferedResult'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/
ForestBufferedresult.xml'));
                                        routeCase();
                                       }
                                }
                               elseif ($taskType=="SERVICETASK")
        {
          $ActivityName=InstantiateTask();// Instiantiate function
                                        if ($ActivityName=="WPS Buffer")
                                        {

$parameter=GetVariables(array('WFS_Admin_Boundary','Distance','Base_Url'));
                                          $WFSBoundary=$parameter[0]->value;
                                                $WFSDistance=$parameter[1]->value;
                                                $WFSBoundary = str_replace('&', ',', $WFSBoundary);
                                                $BaseUrl=$parameter[2]->value;

$StatusID=ExecuteService(array('$WFSBoundary','$WFSDistance',' $BaseUrl'));
                                                if ($StatusID==200)
                                                {
                                                 NextTask();
                                                }
                                       }
                                       if ($ActivityName=="WPS Intersect admin boundary and Forest")
                                       {

$parameter=GetVariables(array('WPSBuffer_Result','WFSForest','Base_Url'));
                                         $BufferResult=$parameter[0]->value;
                                               $WFSForestUrl= $parameter[1]->value;
                                               $WFSForestUrl = str_replace('&', ',', $WFSForestUrl);
                                               $BaseUrl= $parameter[2]->value;

        $StatusID=ExecuteService(array('$BufferResult','$WFSForestUrl',' $BaseUrl'));
                                               if ($StatusID==200)
                                                {
                                                  NextTask();
                                                }
            }
                                       if ($ActivityName=="WPS Buffer Forest")
                                       {

$parameter=GetVariables(array('WPSForestBufferedResult','BuiltupFeaures','Base_Url'));
                                          $WPSForestresult=$parameter[0]->value;
                                               $WFSBuiltupFeauresUrl= $parameter[1]->value;
                                               $WFSBuiltupFeauresUrl = str_replace('&', ',',
$WFSBuiltupFeauresUrl);
                                               $BaseUrl=$parameter[2]->value;
```

```
        $StatusID=ExecuteService(array('$WPSForestresult','$WFSBuiltupFeauresUrl',' $BaseUrl'));
                                            if ($StatusID==200)
                                             {
                                               NextTask();
                                             }
            }
                                        if ($ActivityName=="WPS Intersect Forest and Built-up")
                                        {

$parameter=GetVariables(array('WPSIntersectionResult','Distance','Base_Url'));
                                          $WPSForestresult=$variablesArray[0]->value;
                                                $WFSDistance=$variablesArray[1]->value;
                                                $BaseUrl=$variablesArray[2]->value;

        $StatusID=ExecuteService(array('$WPSForestresult','$WFSDistance',' $BaseUrl'));
                                            if ($StatusID==200)
                                             {
                                               NextTask();
                                             }
            }
                                      }
                                elseif ($taskType=="EMPTY")
        {

SendResult(array('Final_Result'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/FinalResult.x
ml'));
                                  routeCase();
                                }

       else
          {
            print "sorry no task to execute";
          }
       break;

…….. // up to case 12



              case 12:
        $taskType=CheckTaskType();
          //Check the returned task;
                                if ($taskType=="USERTASK")
        {
                                 $ActivityName=InstantiateTask();
                                 if ($ActivityName=="WFS Forest")
                                 {
                                   routeCase();//move to the next task                          }
                                 if ($ActivityName=="WFS Built-Up Area")
                                 {
                                   routeCase();//move to the next task
                                 }
                                 if ($ActivityName=="WFS Admin boundary and Distance")
                                 {
```

```
                routeCase();//move to the next task
               }
               }
               elseif ($taskType=="RECEIVETASK")
        {
                $ActivityName=InstantiateTask();//check the name of activity
                        if ($ActivityName=="Buffered Admin Boundary")
                        {

SendResult(array('WPSBuffer_Result'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/Boundaryresult.xml'));
                         routeCase();
                        }
                        if ($ActivityName=="Intersected Area")
                        {

SendResult(array('WPSIntersectionResult'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/Forest_boundaryIntersection.xml'));
                         routeCase();
                        }
                        if ($ActivityName=="Buffered Forest")
                        {

SendResult(array('WPSForestBufferedResult'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/ForestBufferedresult.xml'));
                         routeCase();
                        }
               }
               elseif ($taskType=="SERVICETASK")
        {
          $ActivityName=InstantiateTask();// Instiantiate function
                         if ($ActivityName=="WPS Buffer")
                         {

$parameter=GetVariables(array('WFS_Admin_Boundary','Distance','Base_Url'));
                          $WFSBoundary=$parameter[0]->value;
                                $WFSDistance=$parameter[1]->value;
                                $WFSBoundary = str_replace('&', ',', $WFSBoundary);
                                $BaseUrl=$parameter[2]->value;

$StatusID=ExecuteService(array('$WFSBoundary','$WFSDistance',' $BaseUrl'));
                                if ($StatusID==200)
                                {
                                 NextTask();
                                }
                         }
                         if ($ActivityName=="WPS Intersect admin boundary and Forest")
                         {

$parameter=GetVariables(array('WPSBuffer_Result','WFSForest','Base_Url'));
                          $BufferResult=$parameter[0]->value;
                               $WFSForestUrl= $parameter[1]->value;
                               $WFSForestUrl = str_replace('&', ',', $WFSForestUrl);
                               $BaseUrl= $parameter[2]->value;
```

```
        $StatusID=ExecuteService(array('$BufferResult','$WFSForestUrl',' $BaseUrl'));
                                                if ($StatusID==200)
                                                 {
                                                   NextTask();
                                                 }
            }
                                        if ($ActivityName=="WPS Buffer Forest")
                                        {

$parameter=GetVariables(array('WPSForestBufferedResult','BuiltupFeaures','Base_Url'));
                                          $WPSForestresult=$parameter[0]->value;
                                                $WFSBuiltupFeauresUrl= $parameter[1]->value;
                                                $WFSBuiltupFeauresUrl = str_replace('&', ',',
$WFSBuiltupFeauresUrl);
                                                $BaseUrl=$parameter[2]->value;

        $StatusID=ExecuteService(array('$WPSForestresult','$WFSBuiltupFeauresUrl',' $BaseUrl'));
                                                if ($StatusID==200)
                                                 {
                                                   NextTask();
                                                 }
            }
                                        if ($ActivityName=="WPS Intersect Forest and Built-up")
                                        {

$parameter=GetVariables(array('WPSIntersectionResult','Distance','Base_Url'));
                                          $WPSForestresult=$variablesArray[0]->value;
                                                $WFSDistance=$variablesArray[1]->value;
                                                $BaseUrl=$variablesArray[2]->value;

        $StatusID=ExecuteService(array('$WPSForestresult','$WFSDistance',' $BaseUrl'));
                                                if ($StatusID==200)
                                                 {
                                                   NextTask();
                                                 }
            }
                                    }
                                elseif ($taskType=="EMPTY")
        {

SendResult(array('Final_Result'=>'http://win371.ad.utwente.nl/student/s6019978/Thesis/FinalResult.x
ml'));
                                  routeCase();
                                }

         else
           {
             print "sorry no task to execute";
           }
        break;
                    default:
                                        //print 'default';
                                  print 'Sorry the process is ended';
       }
```

ANNEX 3: Landuse Planning Result Visualization Codes

```
<!DOCTYPE html>
<html>
  <head>
    <title>WPS Orchestration result visualization</title>
    <link rel="stylesheet" href="openlayers/theme/default/style.css" type="text/css">
    <link rel="stylesheet" href="default.css" type="text/css">
    <link rel="stylesheet" href="style.css" type="text/css">
                <link rel="stylesheet" href="main.css" type="text/css">
    <script src="OpenLayers/OpenLayers.js"></script>
                <script src="http://maps.google.com/maps/api/js?v=3&amp;sensor=false"></script>
                  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
                <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-
scale=1.0, user-scalable=0">

  </head>

    <h1></h1>


    <script type="text/javascript">
          function init() {
       var map = new OpenLayers.Map("map", {
         projection: new OpenLayers.Projection("EPSG:3857"),
       });
                            var style2 = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']);
                            style2.fillOpacity = 0.2;
                            style2.graphicOpacity = 2;
                            style2.strokeWidth = 4;
                            style2.strokeColor = "Yellow";
                            style2.strokeOpacity = 2;


                            var style = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']);
                            style.fillOpacity = 0.2;
                            style.graphicOpacity =2;
                            style.strokeWidth = 4;
                            style.strokeColor = "Green";
                            style.strokeOpacity = 2;
                            style.backgroundGraphic= "Green";

                            var style3 = OpenLayers.Util.extend({},
OpenLayers.Feature.Vector.style['default']);
                            style3.fillOpacity = 0.2;
                            style3.graphicOpacity = 1;
                            style3.strokeWidth = 4;
                            style3.strokeColor = "MediumPurple";
                            style3.strokeOpacity = 0.5;
                            style3.backgroundGraphic= "MediumPurple";
```

```
                        gmap = new OpenLayers.Layer.Google("Google Streets");
        var base = new OpenLayers.Layer.OSM("OSM Map");
        map.addLayer(gmap);
        map.addLayer(base);
        var center = new OpenLayers.LonLat(767373.67735,6840176.77234);
        map.setCenter(center,12);
        //map.addControl(new OpenLayers.Control.OverviewMap());
        map.addControl(new OpenLayers.Control.MousePosition());
        map.addControl(new OpenLayers.Control.LayerSwitcher());

              map.addControl(new OpenLayers.Control.KeyboardDefaults());


                        var Features = new OpenLayers.Layer.Vector("Enschede", {
        strategies: [new OpenLayers.Strategy.BBOX()],
         protocol: new OpenLayers.Protocol.WFS({
                         version: "1.0.0",
                         url: "http://win371.ad.utwente.nl/cgi-
bin/mapserv.exe?map=//win371.ad.utwente.nl/student/s6019978/Thesis/configWFS.map",
                         featureType: "neighbourhood",
                         geometryName: "geom",
                         srsName: "EPSG:3857"

                                 }),
                                 styleMap: new OpenLayers.StyleMap({
            strokeWidth: 4,
            strokeColor: "red",
                                 fillColor: "green",
                                  fillOpacity: 0.2,
                                  backgroundGraphic:"Yellow"
           }),

                        });
                        //console.log(Features)
                        map.addLayer(Features);

                        var BuiltUpFeatures = new OpenLayers.Layer.Vector("Enschede", {
        strategies: [new OpenLayers.Strategy.BBOX()],
         protocol: new OpenLayers.Protocol.WFS({
                         version: "1.0.0",
                         url: "http://win371.ad.utwente.nl/cgi-
bin/mapserv.exe?map=//win371.ad.utwente.nl/student/s6019978/Thesis/configWFS.map",
                         featureType: "urban_areas",
                         geometryName: "geom",
                         srsName: "EPSG:3857"

                                 }),
                                 styleMap: new OpenLayers.StyleMap({
            strokeWidth: 2,
            strokeColor: "red",
                                 fillColor: "blue",
                                  fillOpacity: 0.2,
                                  //backgroundGraphic:"khaki"
           }),
```

```
                        });
                        //console.log(Features)
                        //map.addLayer(BuiltUpFeatures);


         var WPSIntersection = new OpenLayers.Layer.Vector("WPSIntersection", {
                    style : style,
                strategies: [new OpenLayers.Strategy.Fixed()],
                protocol: new OpenLayers.Protocol.HTTP({
                  url: "Forest_boundaryIntersection.xml",
                  format: new OpenLayers.Format.GML()
                })
});
        map.addLayer(WPSIntersection);
         var WPSBuiltupIntersection = new OpenLayers.Layer.Vector("WPSBuiltupIntersection", {
                    style : style3,
                strategies: [new OpenLayers.Strategy.Fixed()],
                protocol: new OpenLayers.Protocol.HTTP({
                  url: "FinalResult.xml",
                                                        styles: 'greenline',
                  format: new OpenLayers.Format.GML()
                })
});
        map.addLayer(WPSBuiltupIntersection);
        var WPSBuffer = new OpenLayers.Layer.Vector("WPSBuffer", {
                   style : style2,
                strategies: [new OpenLayers.Strategy.Fixed()],
                protocol: new OpenLayers.Protocol.HTTP({
                  url: "BoundaryBufferresult.xml",
                  format: new OpenLayers.Format.GML()
                })
});
       // map.addLayer(WPSBuffer);

        var WPSBufferedForest = new OpenLayers.Layer.Vector("WPSBufferedForest", {
                   style : style2,
                strategies: [new OpenLayers.Strategy.Fixed()],
                protocol: new OpenLayers.Protocol.HTTP({
                  url: "ForestBufferedresult.xml",
                  format: new OpenLayers.Format.GML()
                })
});
        map.addLayer(WPSBufferedForest);



                        map.addControl(new OpenLayers.Control.Attribution());
        map.addControl(new OpenLayers.Control.Navigation());  //Adds zoom & drag functionality to
the map
        map.addControl(new OpenLayers.Control.PanZoomBar());   //Displays the zoom in & zoom out
controls
        var scaleline = new OpenLayers.Control.ScaleLine({
           div: document.getElementById("scaleline-id")
        });
        map.addControl(scaleline);
```

```
}
      </script>
                  <body onload="init()">


                     <div id="map" style="position: absolute; left: 5px; top
: 5px; width: 510px; height: 700px; overflow: hidden; border
: 1px solid blue; "></div>

<div id="legend" style="position:absolute; width:250px;
height:200px; left: 520px; top: 5px; overflow: auto;
border: 1px solid blue; ">
<hr size="5" align="left" width="10%" color="#FF0000">: Enschede admin boundary<br/>
<hr size="5" align="left" width="10%" color="#9966CC">: boundary of buit-up area<br/>
<hr size="5" align="left" width="10%" color="#008000">: boundary of forest area<br/>
<hr size="5" align="left" width="10%" color="#FFFF31">: boundary of buffered Forest<br/>

</div>

   </body>
</html>
```

ANNEX 4: Python script to send WPS execute request to WPS server

Buffer operation

```
import requests
import sys
import urllib.request
import cgitb; cgitb.enable()
import  cgi
import json
import xml.etree.ElementTree as ET
try:
    tree = ET.ElementTree(file='InputTest.xml')
    WFS_AdminBoundary_Url=sys.argv[1].replace(",","&") #Get Processmaker WFS admin Features
variable in Python
    DistanceBuffer=float(sys.argv[2]) #Get Processmaker Distance variable in Python
    #Base_url = 'http://130.89.236.184:8080/geoserver/ows'
    Base_url=sys.argv[3]
    #print(DistanceBuffer)
    #print(WFS_AdminBoundary_Url)
    #Base_url = 'http://130.89.236.184:8080/geoserver/ows'
    form = cgi.FieldStorage()
    root = tree.getroot()
  ###################################################################
    for child in root.iter():
        #print(child.tag,child.attrib,child.text)
        if ((child.tag=='{http://www.opengis.net/wps/1.0.0}LiteralData')and (child.text is None)):
            child.text=str(DistanceBuffer)
        if child.tag=='{http://www.opengis.net/wps/1.0.0}Reference':
            textvalue=child.attrib
            if not textvalue['{http://www.w3.org/1999/xlink}href']:
                textvalue['{http://www.w3.org/1999/xlink}href']=WFS_AdminBoundary_Url
    tree.write('output.xml')
  ###################################################################
    file=open('output.xml','r')
    k=file.read()
    headers={'Content-Type': 'application/xml'}
    # Send WPS execute operation to the Server
    resp=requests.post(Base_url,data=k,headers=headers)
    print(resp.status_code)
    file=open('BoundaryBufferresult.xml','w')
    file.write(resp.text)
    file.close()
except Exception as e:
    print (str(e))
```

Intersection Operation

```
import requests
import cgitb; cgitb.enable()
import sys
import xml.etree.ElementTree as ET

Buffer_Result=sys.argv[1] #Get WPS buffer resuli from processmaker variable to Python
```

```
WFS_Forest_Url= sys.argv[2].replace(",","&") #Get Processmaker WFS Forest Features variable in
Python
Base_url=sys.argv[3]
try:
   #Base_url = 'http://130.89.236.184:8080/geoserver/ows'
   #def Intersection_Opeartion(Base_url,WFS_Forest_Url,Buffer_Result):
   tree = ET.ElementTree(file='IntersectionInputdata.xml')
   root = tree.getroot()

#################################################################
   for child in root.iter():
      if child.tag=='{http://www.opengis.net/wps/1.0.0}Reference':
         textvalue=child.attrib
         if textvalue['{http://www.w3.org/1999/xlink}href']:
            textvalue['{http://www.w3.org/1999/xlink}href']=Buffer_Result

         if not textvalue['{http://www.w3.org/1999/xlink}href']:
            textvalue['{http://www.w3.org/1999/xlink}href']=WFS_Forest_Url

   tree.write('output.xml')

#################################################################
   file=open('output.xml','r')
   k=file.read()
   headers={'Content-Type': 'application/xml'}
   # Send WPS execute operation to the Server
   resp=requests.post(Base_url,data=k,headers=headers)
   print(resp.status_code)
   file=open('Forest_boundaryIntersection.xml','w')
   file.write(resp.text)
   file.close()

except Exception as e:
   print (str(e))
```

ANNEX 5: XML File Used As Data Input To WPS Server

Buffer

```
<?xml version="1.0" encoding="UTF-8"?><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.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>gs:BufferFeatureCollection</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>features</ows:Identifier>
      <wps:Reference mimeType="application/wfs-collection-1.0" xlink:href="WFS URL"
method="GET"/>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>distance</ows:Identifier>
      <wps:Data>
        <wps:LiteralData>Buffer Distance</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="text/xml; subtype=wfs-collection/1.0">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```

Intersection

```
<?xml version="1.0" encoding="UTF-8"?><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.opengis.net/gml" xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:wcs="http://www.opengis.net/wcs/1.1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsAll.xsd">
  <ows:Identifier>vec:IntersectionFeatureCollection</ows:Identifier>
  <wps:DataInputs>
    <wps:Input>
      <ows:Identifier>first feature collection</ows:Identifier>
      <wps:Reference xlink:href="BufferResult URL" method="GET"/>
    </wps:Input>
    <wps:Input>
      <ows:Identifier>second feature collection</ows:Identifier>
      <wps:Reference xlink:href="Feature WFS URL" method="GET"/>
```

```
    </wps:Input>
  </wps:DataInputs>
  <wps:ResponseForm>
    <wps:RawDataOutput mimeType="application/wfs-collection-1.0">
      <ows:Identifier>result</ows:Identifier>
    </wps:RawDataOutput>
  </wps:ResponseForm>
</wps:Execute>
```