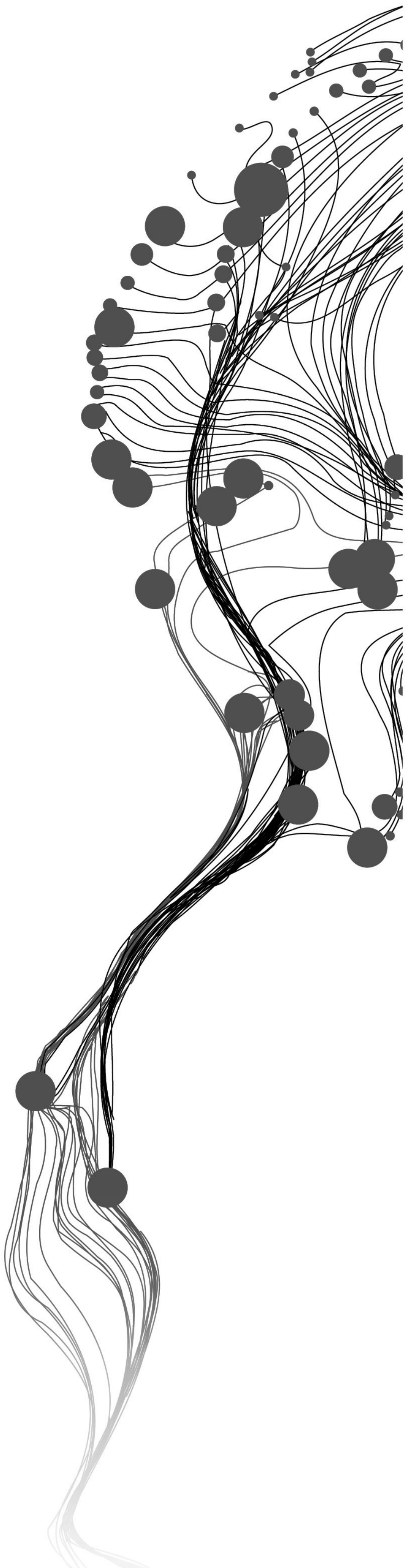
DESIGN AND ORCHESTRATION OF WEB PROCESSING SERVICES AS SERVICE CHAINS

MEAZANESH ASRES ALEMU
March, 2012

SUPERVISORS:

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



DESIGN AND ORCHESTRATION OF WEB PROCESSING SERVICES AS SERVICE CHAINS

MEAZANESH ASRES ALEMU
Enschede, The Netherlands, March, 2012

ABSTRACT

Currently, several OGC WPS processing-functionalities have been implemented and exposed as service over the web. These WPS processing-functionalities are already providing a useful application to generate geospatail information out of geospatail data on-the-fly in a distributed environment. However, individual processing-functionalities are not sufficient to solve a complex geoprocessing task, as the complex nature of geospatail data often requires multiple geoprocessing-steps. Therefore, chaining of individual WPS processing-functionalities to buildup larger application that can solve complex geoprocessing task is necessary. The current popular standard, BPEL, to orchestrate services into service chain, which is dependent on WSDL, is not suitable for geoservices. The WPS processing-functionalities are not described by WSDL mandatorily, instead using *DescribeProcess* in detail, and using *Getcapabilities* briefly. Consequently, it is mandatory to generate WSDL for each processing-functionalities to be orchestrated in BPEL, taking up much time and effort. This requirement limits their application to be utilized through chaining. Therefore, it is necessary to devise a method to chain without extra description requirements per processing-functionalities.

This research was an attempt to provide a method for chaining of disparate processing- functionalities to be utilized as single application through chaining without the need to generate WSDL per each processing- functionaries to support expert users in GI application domain. For this we designed a generic chaining schema using XSD language. The chaining schema is generic by defining all possible contents required for chaining conceptually, serving as template, to chain any two or more number of processing-functionalities in any GI application domain. This chaining schema basis on selected parameters of *DescribeProcess* response content. In addition, architecture for chaining of processing-functionalities using the designed chaining schema is provided. A proof-of-concept was demonstrated with implementation of the designed chaining schema using a real-world application scenario.

Keywords

web services, geoservices, web processing service, processing-functionalities, orchestration, chaining engine, service chain, chaining schema

ACKNOWLEDGEMENTS

This is a great opportunity to express my deep and sincere gratefulness to my first supervisor, Dr.J.M (Javier) Morales, for his unlimited time for discussion and his patience in guiding me throughout this research period. I also would like to thank my second supervisor, Dr.Ir. R.L.G.(Rob) Lemmens, for sharing his idea and support.
I also gratefully acknowledge the financial support that I received from the Government of the Netherlands through NUFFIC organization, and ITC for giving me the opportunity to follow this Master of Science in GFM.

My thanks also goes to my GFM classmates and all other friends for their encouragement and friendship throughout this study.
Special thanks to my beloved families and other friends(Mulluken, Tadesse, Dagne, and Mamaru) for thier continues encouragement.
Above all, I would like to thank the almighty God for everything he is doing in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1
Introduction

1.1 MOTIVATION AND PROBLEM STATEMENT

Geospatial information (GI) has been playing an important role in almost all walks of our life. GI became so important due to its natural capability of holding location referenced information, providing deep understanding of the geospatial phenomenon that occurs around us. GI has been used for answering from simple to complex geospatail questions. Historically, GIs have been extracted, acquired, stored, analyzed, and visualized using tightly-coupled traditional GIS (Geographical Information System) systems [21].

With the recent advancement in SOA (Service Oriented Architecture), the traditional GIS is moving towards highly distributed web service-based application environment. SOA is "a paradigm for organizing and utilizing distributed application capabilities that may be under the control of different ownership domains" [22]. Web services are modular applications that can be published, located, invoked, and can be combined to form complex application across the web [28].

SOA's basic idea is separating functionalities of tightly-coupled application, like traditional GIS, into loosely-coupled distinct services. So that application owners make services accessible over the web to allow users to build-up new application by orchestration of individual web services. Orchestration is the process of combining and coordinating a set of existing web services to create new application requirements [27].

Web services play a key role for publishing and discovering GI resources such as geospatial data and processing tools in distributed environments using distributed servers. Similar to other application fields, GI application fields got many advantages in different aspects from web services. For example, if organizations face a shortage of information to answer geospatial questions, when there is shortage of money to acquire new geospatial data, when accessibility is not possible to get new geospatial data, and to avoid more effort for acquisition of new geospatial data.

Along the same line of SOA, OGC (Open Geospatial Consortium) provides geospatial services such as WMS (Web Mapping Service), CSW (Catalog Services for the Web), WFS (Web Feature Service), and recently WPS (Web Processing Service). Geospatial services are web services which are specialized to smooth sharing, accessing, and processing of GI resources among organization, private companies, and individuals over the web.The geospatial service is commonly abbreviated as geoservice and we used this term in the rest of this thesis.

Today, having those roads of sharing environments, an increasing number of organizations, companies, and individuals are providing their GI resources as geoservices over the web. Yet it is usually not sufficient to use only an isolated geoservice to solve a real-world geospatial problem. To utilize those geoservices, users have to locate relevant geospatial data, assemble geospatial data from several sources, process the geospatial data and then visualize the results.

In most cases, users are supported by CSW to find relevant geospatial data, WMS to visualize the result, and recently WPS allows access to remote GIS proccessing functionalities (e.g. buffer). Sometimes users can create simple service chains by configuring manually to analyze geospatial data. But, these service chains are not very flexible and only applicable for predefined tasks. Consequently, a user has to download a huge amount of geospatial data and analyze with traditional GIS applications [15] [14]. In addition, most GI applications require a long iterative processing steps, reformatting, and integrating various types of geospatial data to answer complex questions. This kind of processing is laborious and requires intensive knowledge of processing steps [18] [17]. In addition, to upgrade functionality of traditional GIS application needs restructuring of the entire system as those functionalities are tightly-coupled. As a result, these functionalities cannot be reused in combination with other applications.

Thus, it is necessary to replace the functionality of traditional GIS application by orchestrating a set of loosely-coupled geoservices, towards SOA, to achieve reusability of functionalities and independent upgrade capability. The GI community is aware of this necessity and several researches carried out using the existing standards and technologies for orchestration of existing geoservices, for example [29]and [26].

The existing common standards and technologies for orchestration of web services is BPEL(Business Process Execution Language) [1] which is dependent on WSDL(Web Service Description Language) [10]. WSDL is a standard for defining web services interfaces hiding the implementation details of web services.

Most geoservies are not described by WSDL. To use BPEL for orchestration of geoservices, the user who design service chain has to define the WSDL for each individual participant geoservices. The same thing will be done for adding new geoservice functionality in the orchestration. As a result, it creates a lot of tasks for orchestration of geoservices using BPEL as described in [29] [26]. In addition to this burden, BPEL is not capable of binary data transmission [29] [26].

Problem Statement

The existing mechanisms, WSDL and BPEL, to design and orchestrate geoservices as service chains are not flexible up to requirements of user who designs service chain.

1.2 RESEARCH IDENTIFICATION

The main issues that are addressed in this proposed research are defined through the following research objectives and research questions.

1.2.1 Research objectives

The main objective of this research is to develop a method to design web processing services as service chains and to orchestrate the service chains. In order to meet the main objective the following four sub objectives should be addressed.

1. To identify the requirements for creating service chain.
2. To choose a method for discovery of services for the service chain.
3. To design an assembly of individual services into a service chain.
4. To provide a mechanism to orchestrate the service chain into an executable workflow.

1.2.2 Research questions

To achieve the objectives, the following research questions need to be answered.

1. What are the requirements to create service chain (for sub objective 1)
2. How to discover services to create service chain? (for sub objective 2)

3. How to assemble a set of services as service chain? (for sub objective 3)

4. How to present the service chain as a single service? (for sub objective 3)

5. How to coordinate or control the interaction among a set of participant services for input/output messages exchange (i.e. how the output of one service can be used as input for the other services)? (For sub object 4)

1.2.3 Innovation aimed at

This research has aims of developing a new method to design web processing services as service chains and to orchestrate the service chains. This method will be applied in any application domain of GI to design web processing service as service chain to accomplish complex tasks that cannot be achieved by a single service.

1.2.4 Related work

Several research works have been done for chaining of distributed services to facilitate access and visualization of geospatial data, although the incorporation of processing services in the service chain is new paradigm. For example, [4] [5] described three types of services chaining types for geoservices called client-coordinated or transparent services chaining, translucent services chaining , and opaque services chaining. Those approaches are mainly deal in hiding the complexity of services chain to the users. [18] [17] presented a methodology for chaining of geoservices that integrates syntactic and semantic services descriptions. Deep service description in terms of both syntactic (structure) and semantic (meaning) is also proposed for services chaining. In this approach, services chaining procedure is decomposed into service discovery, abstract chaining, concrete chaining, and execution of services chains.

Recently, the development of WPS and GPW (GeoProccessing Workflow) [24] motivate the incorporation of geospatial processing capability in chaining of distributed services. GPW is a combination of two concepts geoprocessing (processing of geospatial data) and workflow (automation of process flow). GPW is developed to enhance the flow of processes in chaining of geoservices to achieve distributed service-oriented processing capability. Currently, GPW is under investigation for supporting different capabilities such as how to bring human interaction in the workflow and how to assemble workflow automatically.

Using WPS as solution for chaining of distributed services is also investigated. For example, [20] has evaluated the feasibility of the WPS specification for creating distributing geospatial processing services and they concluded as is it is workable. [26] and [29] also investigated the possibility of using WPS for orchestration of a set of geoservices for disaster management use case. In this approach all participant services are defined by hard coding (configuration) and new services can be added with the same approach. This approach is not flexible since the functionality of service chain is limited to already predefined tasks and needs knowledge of customizing the code. [31] also applied similar approach for extraction of water information use case using distributed services. However, all required data are restricted only to one services provider. Similarly, [12] proposed similar approach for forest risk analysis but different in that all required functionalities (data and processing functionalities) comes from a single service provider. Although, the approach has advantages of increasing performance, it results remote tightly-coupled processing application which is against service reusability and flexibility of service-oriented application.

Even though, those researches presented applicable approaches for chaining of distributed services, there is no method that can be applied for any GI application. Most of the researches are designed for specific problem only. Therefore, this research will provide method for designing and assembly of services that can be applied in any GI application.

1.3 PROJECT SET-UP

In order to achieve our objective, the research questions raised above were answered. The approach for answering those research questions are described below.

1.3.1 Method adopted

1. Literature review: First of all, literatures were studied to get understanding on basic concept of SOA, web services, geoservices. In addition, the current sates-of-the-art of WPS specification were studied and discussed in detail.

2. Service chaining requirement analysis: Subsequent to literature review, analysis on service chaining requirements and service discovery method is done thereof to capture the functional requirements in order to design the chaining method.

3. Chaining method design: Following the requirement analysis of service chaining, the chaining method is designed using XML (Extensible Markup Language) schema (XSD) definition language.

4. Evaluation: Finally, the functionality of the designed chaining method is implemented and demonstration is done for a chosen scenario as proof of concept.

1.4 STRUCTURE OF THE THESIS

The research conducted during this thesis period is structured into 6 chapters, as briefly described below:

Chapter 1 Problem statement and motivation: provides the outline of the research, describing the research motivation, research problem, research objectives, research questions, related works, and method adopted are introduced.

Chapter 2 Web service based architecture: some literature were reviewed on the basic concept of SOA, web services, geoservices, and the detailed sates-of-the-art of WPS specification.

Chapter 3 Service chaining requirement analysis: analysis of service chaining requirements is conduced from general web service and geoservices literatures. The result of this analysis is used as input to design the chaining method for WPS processing-functionalities, in chapter 4.

Chapter 4 Method for chaining of WPS processing-functionalities: provides the generic chaining method to chain WPS processing-functionalities and the architecture for chaining of WPS processing-functionalities using the designed chaining method.

Chapter 5 Prototype implementation and testing: provides the implementation of the chaining method and as proof-of-concept using application scenario.

Chapter 6 Discussion, Conclusion, and Recommendation: The result of the thesis are discussed in relation to the the research questions. The conclusion of the thesis and recommendation for future improvements are also provided.

Chapter 2
Web service based architecture

2.1 INTRODUCTION

Nowadays, there are a number of resources which are designed and provided as web services over the internet. In order to utilize the capabilities which are provided via web services, there should be some architectural design which enables to discover them and combine in order to satisfy more than one needs. SOA (Service oriented Architecture) is one of the architectural design to meet this requirements. SOA based design can be implemented using web services or geospatail web services. In this chapter, basic concepts SOA in section 2.2, Web services in section 2.3, Asynchronous communication in section 2.4, and geoservices in section 2.5 are introduced. The last section 2.6 is investigates the sates-of-the-arts of web processing services(WPS) specification.

2.2 SERVICE ORIENTED ARCHITECTURE(SOA)

As it is defined in OASIS reference model, SOA is "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [22]. These capabilities are created by distributed owners (e.g organizations, private companies, and individuals) as a solution for their business problems. Obviously, these capabilities can also satisfy one or more other user needs either individually or in combination. Before combining the capabilities or using a single capability to satisfy the needs, matching between needs and capabilities is required. Before matching and combining, capabilities and needs should come together virtually by means of a services. A service is a central element of SOA to enable access to one or more capabilities. A service is accessed via a service interface. A service interface is a specification about the service such as service type, inputs and outputs parameters, and functionality by hiding the internal implementation details of the services. These descriptions are required for matching the capabilities against the user needs. The SOA paradigm has a framework to this requirements. The framework, called publish-find-bind paradigm, is comprised of three participants and three basic operations, see figure 2.1. The three participants are service provider, service broker, and service requester.

- Service provider: service providers publish (register) service interfaces to service broker to offer services.
- Service requestor: service requesters find (search) appropriate services from service broker to use for their needs.
- Service broker: service broker is a repository of searchable service descriptions which are provided by service providers. The service broker is used for advertising and locating services bringing the services and service requestors together.

The three SOA participant interact using three basic operations: publish, find, and bind.

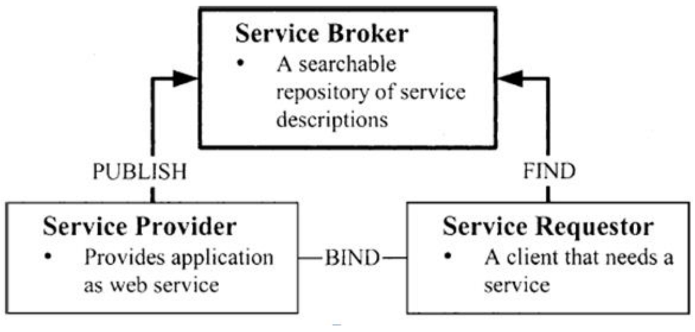

Service Broker
• A searchable repository of service descriptions
PUBLISH
FIND
Service Provider
• Provides application as web service
BIND
Service Requestor
• A client that needs a service

Figure 2.1: SOA Publish-Find-Bind paradigm

- Publish: is the process of advertising (making services description available) on the service broker. A service provider publishes the service descriptions on service broker to be discovered by the service requestors.
- Find: is the process of searching suitable services which are appropriate for service requestor needs. A service requestor provides find criteria such as service type, input and output parameters to the service broker. The service broker searches by matching the find criteria against the published service descriptions. Then, if matches found, the list of matched candidate service will be returned to the service requestor.
- Bind: after the searching process found candidate services and if there are more than one matches are found, the service requestor can choose the most suitable service based on their descriptions. The bind operation allows further communication between the service requestor and the chosen service to use the actual service functionality.

2.3 WEB SERVICES

As SOA is a concept, a mechanism is required to realize it. Web services are one of the most preferred way for realization of SOA in practice [13]. The fundamental web service standards include XML(Extensible Markup Language), HTTP(Hypertext Transfer Protocol), WSDL, SOAP(Simple Object Access Protocol), and UDDI(Universal Description Discovery Integration).

- XML: a standard that provides a way to encode machine and human understandable data format. Every web services standards are designed based on XML, XSD (XML Schema Definition) and XML NameSpaces [13]. XML NameSpaces provides a means to avoid name conflicts between XML elements, guaranteing uniqueness which are defined in XSD.

  XSD (XML Schema Definition), a recommendation of the W3C, specifies how to describe the elements and attributes in an XML document. A schema is a data structuring mechanism which helps to organize and interpret large amount of information easily. The description is used to verify whether each items which occurs in the XML document content is valid or not. Generally, schema is an abstract representation of an object's characteristics and relationship to other objects. Before, creating any schema, analysis of elements structure and their relationship is required. XSD has many advantages over earlier XML schema languages, such as DTD (Document Type Definition). One of the advantages is that XSD is directly written in XML, in contrast to the earlier languages, which implies that XSD doesn't need an intermediate processing step by a parser. In addition, XSD can be queried through XSLT (Extensible Stylesheet Transformations).

- HTTP: is a transport protocol widely used by Internet to send web services. A protocol is an agreement or standardized method for establishing communication between services to transmit data or message, as humans need to have a common language before they start to communicate to each other.
- WSDL [10]: used to describe characteristics of web services. WSDL description includes interface, what service functionality offer including the name of the functionality and their input and output parameters, binding information-how a service can communicate to other service, and services address (URL), where a service resides.
- SOAP [8]: is a standard which defines an XML based messaging exchange protocol to allows one web services to call another web services functionalities, pass input parameters, and receive results.
- UDDI [7]: is a directory of services descriptions which are published by service providers. UDDI enables service consumers to locate candidate services and discover service description by service consumers, so that the user can validate the service fitness for their requirements.

2.4 ASYNCHRONOUS COMMUNICATION

Synchronicity refers to the binding of the service requestor to the execution of the service. In synchronous communication pattern, the service requestor blocked and waits for the service until the execution completed and get response before continuing to another tasks. In contrast to synchronous, asynchronous communication pattern allow a service requestor to invoke a service and then execute other functionalities or any other task. In addition, in asynchronous, the status of execution can be reported to the services requester. In asynchronous, the execution result can be retrieved at a later time, while in synchronous, service requestor receive the result directly when the task is completed.

2.5 GEOSERVICES

Geoservices are web services which are dedicated to deal on geospatail content. Usually, these are divided into three subcategories [4]:

- Data services: geoservices to geospatail data such as WMS, to access to map layers, vector or raster data in JPEG (Joint Photographic Experts Group)or PNG (Portable Network Graphics) format, WCS (Web Coverage Service) which provides access to raster data, and WFS which provides access to vector data, output encoded in Geography Markup Language (GML). GML is an XML based language used for encoding features.
- Processing Services: geoservices to offer operations for processing such as WPS, transformation services.
- Registry or catalogue services: are services supporting registry and discovery of geospatail data and geoservices such as CSW.

The OGC web services are standardized geoservices to facilitate access to geospatail data and geoprocessing algorithms through Internet. The OGC geoservices uses HTTP for transport, XML/GML for message encoding, GetCapabilities operation for service description instead of

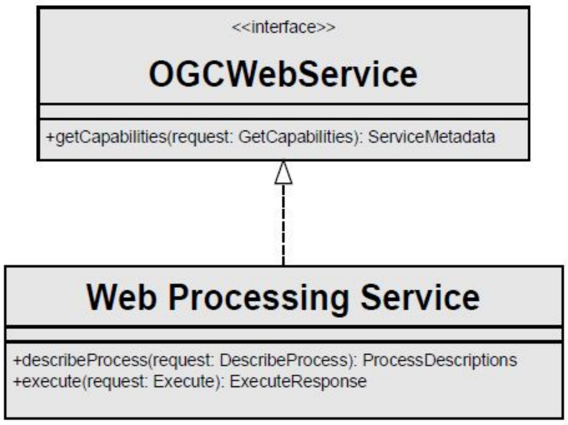

<<interface>>
OGCWebService
+getCapabilities(request: GetCapabilities): ServiceMetadata
Web Processing Service
+describeProcess(request: DescribeProcess): ProcessDescriptions
+execute(request: Execute): ExecuteResponse

Figure 2.2: WPS Interface class diagram

WSDL, and catalogue services for web (CSW) for service repository instead of, UDDI. The GetCapabilities operation is common to all OGC services used to request and return service-level metadata.
As geospatial data discovery through CSW, access through WFS and WCS, and visualization through WMS is covered, access to service-based extraction of geospatail information is being covered by WPS. WPS is new relative to other geoservices such as WFS and WMS, but has multiple advantages such as support for service chaining where the integration of geospatail data from multiple sources is required. For these reasons, nowadays, WPS is a hot research issue of GI community researches as the need for integration of geospatail data and processing service on the fly increases for many GI applications [11]. This research is also dedicated in designing a methods for chaining of WPS functionalities. Before, we start the designing method, in the next section of this chapter the current sates-of-the-arts of WPS specification is discussed in detail.

2.6 WEB PROCESSING SERVICES (WPS)

The WPS [25] specification, current version 1.0.0, defines a standardized interface to expose, access, and use of processing-functionality as service. In this thesis we used the terminology "processing-functionality" which an equivalent meaning to terminology "process" in the specification. Both terminology refers to an individual service provided using WPS specification. These processing functionalities can be geoprocessing algorithms or computational models which are designed to perform a specific geoprocessing tasks. The main concept of WPS is to serve as a container of one or more processing-functionalities which can be accessed and executed over a distributed web environment. For example, intersecting two polygons is a single geoprocessing task which can be handled by an "Intersection" processing-functionality which is provided as service. In this manner, multiple processing-functionalities can be provided in a single WPS server implementation. A WPS server is a computer where the WPS specifications is implemented and provided as services which has its own unique address (base URL) over the web. This base URL is used as the entry point to the WPS services, the processing-functionalities.

2.6.1 WPS operations

The WPS has three standardized mandatory operations, namely *GetCapabilities*, *DescribeProcess*, and *Execute*. These operations enable users to retrieve the capabilities, details, and to request for execution of one or more processing-functionality. The communication, request and response to these operations can be done via HTTP GET-with KVP encoding, HTTP POST-with XML encoding, or SOAP with WSDL, as WPS also provides support to SOAP and WSDL. The diagram which summarizes the three operation is shown in figure 2.2.

1. *GetCapabilities* operation: This operation, which is common to all OGC geoservices, allows a user to request and receive back the metadata (information about service) such as information about service providers name or contact address, and access methods for *DescribeProcess* and *Execute* operations. The *GetCapabilities* request parameters ('service' and 'request') are mandatory parameter to be provided for the WPS server to get a successful response. For more information, the *GetCapabilities* mandatory and optional request parameters with short description of each parameters are provided in table 2.1. The *GetCapabilities* operation request can be encoded either in KVP (mandatory) via HTTP Get or in XML (optional) via HTTP Post.

   The response document also includes the list of processing-functionality with brief abstract and keyword description which are available in the requested WPS. Each processing-functionality in the *GetCapabilities* response document has a unique identifier, used to identify a functionality within a single WPS. The *GetCapabilities*response document does not include detail information about individual processing-functionality, such as expected inputs and outputs types by the processing-functionality. To get the full description of each processing-functionality, including their expected inputs and outputs, the *DescribeProcess* operation needs to be requested. For mandatory and optional response parameters of *GetCapabilities*, see table 2.2.

2. *DescribeProcess* operation: This operation allows a user to request and receive back full description about one or more processing-functionality which are listed in the *GetCapabilities* document. The *DescribeProcess* request is based on the processing-functionality identifier (identifier within a single WPS) found in the *GetCapabilities* response. For more information, the *DescribeProcess* mandatory and optional request parameters with brief description are provided in table 2.3. The *DescribeProcess* operation request can be encoded either in KVP (mandatory) via HTTP Get or in XML (optional) via HTTP Post.

   The *DescribeProcess* response document contains the detail information about the requested processing-functionality including required inputs and returned outputs. Each processing-functionality can have any number of inputs and output parameters. The response document also includes whether output storage is supported or not, whether status report is supported or not, input and output data types (complex, literal, Bounding box), allowed maximum complex input size (e.g maximum megabyte of raster image), and others information. For more information, the *DescribeProcess* mandatory and optional response parameters with brief description is provided in table 2.4.

3. *Execute* operation: This operation allows a user to deliver applicable input parameters to the selected processing-functionality of WPS to execute the actual required task and receive back execution result. In contrast to the *Getcapabilities* and *DescribeProcess* operations, the request for *Execute* operation can be encoded either in KVP (optional) via HTTP Get or in XML (mandatory) via HTTP Post.

   The necessary parameters for the *Execute* request including the name of the processing-functionality, applicable inputs, and way of input delivery should be based on the information in *DescribeProcess* and *Getcapabilities* response documents, the mandatory and optional request parameters for execute are provided in table 2.5 on page 16. The inputs can be delivered either directly embedded to the *Execute* request or indirectly by embedding the URLs of web accessible resources. According to the WPS specification, the delivery of inputs in both combination (directly as well as indirectly) with a single execute request is not supported, i.e it is possible to use only one of the two ways at a time. In addition, as stated by the specification, "The normal way to provide large inputs to a WPS is through providing one or more URIs (usually URLs) of input values, unless the inputs are simple scalar values. This is not intended to be used to facilitate batch processing (e.g. multiple images to be processed through a single algorithm). If a process [1]is to be run multiple times (probably using different inputs each time), each run shall be submitted as a separate operation request". This implies, the user cannot provide nested inputs to be executed in iterative way.

The *Execute* operation output can be returned either directly to the user or indirectly can be stored at web accessible location where the URL for the location of output is included in the *Execute* response document. The former option, the directly output return, can be either in raw output (without XML wrapper) for single output or encoded in XML (with XML wrapper). The later option, indirectly return, is possible if the output storage is supported by the WPS server (i.e.*StoreSupported=true* in the *DescribeProcess* response document). For further information, the mandatory and optional execute response parameters are provided in table 2.6 on page 17.

Moreover, the execute operation allows to monitor the on going progress of the task execution via status report element if the status report is supported by WPS server i.e *Status-Supported=true* in the *DescribeProcess* response document. If the status report is supported, then the WPS server will keep the status element of the stored Execute response document up-to-date while the task execution is on running. Therefore, the user can retrieve by polling the updated *Execute* processing-functionality response via the URL included for this purpose in the *Execute* response document. However, the status report is possible if and only if the output storage is supported by WPS server. Furthermore, the *Execute* response document includes information such as process status, list of used inputs, list of outputs or URL to outputs. The process status may be either *ProcessAccepted* which indicates that the task is received and is in queue to be executed, or *ProcessStarted* which indicates that the task is on running, or *ProcessPaused* which indicates that the task execution is paused, or *ProcessSucceeded* which indicates the task execution is completed, or *ProcessFailed* indicating that a problem is occurred. For further information see table 2.7 on page 18.

2.6.2 WPS input and output data types

An input is data provided to processing-functionality to generate output. Output is the result returned after the processing-functionality are executed. The are three kind of data types supported by WPS: Complex data type, literal data type, and bounding box data type.

- Literal data type: Indicates simple textual represented in integer (e.g buffer distance), string, float, general number, measurements such as buffer distance, unit of measure (e.g.meter)
- Complex data type: This data type includes more complex data than simple textual representation, such as coordinated of a polygon encoded in GML, raster image, and URL of GetFeature request for WFS which is delivered as input for execute operation request.
- Bounding box data type: This data type is the upper and lower coordinates of supported spatial reference system.

[1]the word "process" is equivalent to "processing functionality" in our case which both refers to a single WPS functionality within a single WPS

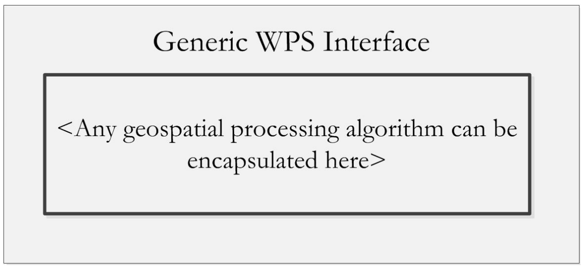

Generic WPS Interface
<Any geospatial processing algorithm can be encapsulated here>

Figure 2.3: WPS wrapping existing algorithms diagram

Table 2.1: WPS GetCapabilities operation request parameters and their description

Name	Multiplicity [2]	Description
Service	1	identifies the requested service type and the value must be "WPS"
Request	1	identifies the requested operation type and the value must be "GetCapabilities"
AcceptVersions	0..1	Identifies the version of WPS specification and the value must be 1.0.0 for the case of WPS version 1.0.0. Used to check the excepted parameters of the requested operation
Language	0..1	Used to specify the response document in human -readable language in case of the WPS implementation support in multiple human languages

2.6.3 WPS for wrapping

WPS is designed to provide a generic interface. As result any existing geospatail algorithm can be wrapped by the generic WPS. The WPS processing-functionality serve as standalone services if all the required input data are provided directly with the execute request. The WPS can be used as middleware for data (when the required input data to be executed by a processing-functionality is provided as web accessible resource), and middleware for software (when any existing geospatail algorithm is wrapped by the WPS interface, see figure 2.3 on page 13).

2.6.4 The need for chaining of WPS processing functionalities

Normally, WPS processing-functionality are atomic service designed to perform specific single geoprocessing task. Although, the atomic processing-functionalities are suitable in supporting flexibility and reusability in different application domain, they are insufficient to perform complex geoproccessing tasks. On the other hand, according to the WPS specification, a single WPS processing-functionality can be designed to perform complex geoproccessing task, which is not flexible to reuse for other application. For this reason, combining a set of atomic processing-functionalities to perform complex geoprocessing task is identified as a requirement. This requirement is identified as one of the required geoprocessing research agenda in [9]. This requirement can be achieved through chaining of individual processing-functionality.

Table 2.2: WPS GetCapabilities operation response parameters and their description

Name	Multiplicity [3]	Description
service	1	identifies the requested service type and the value must be "WPS"
version	1	WPS specification version.Should be 1.0.0 for the case of WSP version 1.0.0
updateSequence	0..1	the version of the service metadata within a single WPS.The version values may be different as they are selected by individual service provider
lang	1	language identifier
ServiceIdentification	1	metadata (capabilities) of the WPS server.Should include title of WPS with narrative description and supported WPS specification versions
ServiceProvider	1	describes the provider of the WPS e.g.organization,company's information such as name and contact address
OperationsMetadata	1	Information about the operations implemented the WPS server including the access methods(HTTP GET, or POST, or both) URLs for each of the three operations
ProcessOfferings	1	contains unordered lists of all available processing-functionality offered by this WPS.Does not include input and output details
Languages	1	list default and supported languages supported by the WPS server
WSDL	0..1	URL of WSDL document describing to the three operations to be retrieved

Table 2.3: WPS DescribeProcess operation request parameters and their description

Name	Multiplicity [4]	Description
Service	1	Identifies service type and the value must be "WPS"
Request	1	Identifies requested operation and the value must be "DescribeProcess"
Version	1	Identifies service specification version. The value must be 1.0.0 for case of WPS version 1.0.0
Language	0..1	Specifies the human readable language of the response document (e.g."en-CA") supported by the WPS server. The language must be listed in the GetCapabilities document
Identifier	1..*	Unordered list of one or more processing-functionality identifiers, separated by commas, for which the description is requested. The processing-functionality identifiers must be listed the GetCapabilities response under ProcessOffering section

Table 2.4: WPS DescribeProcess operation response parameters and their description

Name	Multiplicity	Description
ProcessDescription	1..*	Full description of each requested processing-functionality including list of supported inputs and outputs parameters, whether output storage and status report are supported or not
service	1	Identifies the service type and the value should contain "WPS"
version	1	Indicates the WPS specification version
lang	1	Includes identifier of supported language
ProcessDescription		
Identifier	1	Unambiguous identifier of a functionality, unique for within a single WPS
Title	1	title of a processing-functionality normally available for human use
Abstract	0..1	brief narrative description of processing-functionality available for human use
Metadata	0..*	references to more metadata about the functionality
Profile	0..*	profile to which the WPS processing-functionality complies
processVersion	1	release version of the processing-functionality within a single WPS (not the WPS specification).The value may be different since it is selected by individual service provider
WSDL	0..*	the URL for location of WSDL that describes each processing functionality
DataInputs	0..1	list of mandatory and optional expected inputs by processing-functionality. This section includes. Input Identifier, minimum and maximum occurrence of inputs, and supported input data types
ProcessOutputs	1	List of mandatory and optional supported outputs from the execution of processing-functionality. This section includes outputs identifier, output data type, and supported output delivery mechanisms
StoreSupported	0..1	indicates if one or more complex data output can be stored by WPS server as web accessible resources. The default value is false which means the output must be returned directly encoded in execute response either in raw data if single output or wrapped by XML
StatusSupported	0..1	indicates if status report is supported or not. It is supported if and only if the output storage is supported. If the value is true, then WPS server will keep the status element of the stored execute response document up-to-date while the request execution is running. So that the user can monitor the progress by polling the latest via the URL in the response document. The default value is false means no progress monitoring

Table 2.5: WPS Execute operation request parameters and their description

Name	Multiplicity	Description
Service	1	Identifies service type and the value must be "WPS"
Request	1	Identifies the requested operation type and the value must be "Execute"
Version	1	Indicates WPS specification version and the value must be 1.0.0 for the case of WPS version 1.0.0
Identifier	1	The unique identifier of the processing functionality (identifier of processing functionality within a single WPS). The identifier must be listed in the GetCapabilities response document under "Processoffering" section.e.g Intersection
DataInputs	0..1	List of inputs provided for this processing-functionality execution (e.g. Intersection. At least one input is required unless all the required inputs are located in predetermined fixed resources. Inputs can be provided either by embedding directly with execute request, or by reference as web accessible resources (embedding the URL with the execute request)
ResponseForm	0..1	Used to determine the return type of output depending on the value of "StoreSupported" in the DescribeProcess response whether the storage is supported or not. There three options: Direct return in raw output form if the output is single, direct return wrapped by XML, by reference (URL) if the value of "StoreSupported" is "true" in the DescribeProcess response
Language	0..1	Language identifier and the value must be listed in the GetCapabilities

Table 2.6: WPS Execute operation response parameters and their description

Name	Multiplicity	Description
Service	1	Identifies service type and the value shall contain "WPS"
version	1	Indicates WPS specification version and the value shall be 1.0.0 for the case of WPS version 1.1.0
lang	1	language Identifier
statusLocation	0..1	URL to location where current "ExecuteResponse" document is stored to either to monitor the status of the execution or to retrieve the final output. This is included when "storeExecuteResponse"= "true" in the DescribeProcess response document.
serviceInstance	1	The GetCapabilities URL of this specific WPS which was invoked
Process	1	This description the processing-functionality which did the task including it is identifier, processVersion
Status	0..1	Contains the status of the execution of the given task whether is completed, or on progress, or paused,see table 2.7 on page 18
DataInputs	0..1	list of inputs which were provided to this processing-functionality execution
OutputDefinitions	0..1	list of definitions of outputs requested from executing this processing-functionality
ProcessOutputs	0..1	List of values of outputs from functionality execution. The output return type is based on the value of "ResponseForm" in the execute request. i.e. output may be either raw output or by reference or wrapped by XML

2.7 SUMMARY

In this chapter,we have explored the basic concept of web services and geoservices particularly, in relation with SOA. We have identified usefulness of atomic processing-functionalities to enabling standardized service-based extraction of geospatail data. The next required step is the chaining of individual processing-functionalities to perform complex geoprocessing task.

Table 2.7: WPS Execute operation status codes parameters and their description

Name	Multiplicity	Description
CreationTime	1	The time (UTC) that the task execution finished. If the task execution is not completed , this attribute shall contain the creation time of this document.
ProcessAccepted	0..1	indicates that task has been accepted by WPS server, but is in a queue and has not yet started to execute
ProcessStarted	0..1	indicates that task has been accepted by WPS server, and execution has started.This may include the progress task execution percentage such *X%* of task is completed
ProcessPaused	0..1	indicates that the server has paused the task execution
ProcessSucceeded	0..1	indicates that task execution is successfully completed
ProcessFailed	0..1	indicates that task execution has failed and includes error information

Chapter 3
Service chaining requirement analysis

3.1 INTRODUCTION

The objective of this chapter is to analyze the functional requirements which are needed for chain of two or more number of services from literatures of general information technology (IT) stream web services chaining as well as GI application domain geoservices chaining. The analysis of these requirements is important for this research as we can capture the required information to be adopted to design our chaining method for WPS processing-functionalities in chapter 4, which is the target of this research.

This chapter starts with service chain concepts in section 3.2, service chaining design types in section 3.3, service chaining procedure in section 3.4, and last section 3.5 is summary.

3.2 SERVICE CHAIN CONCEPT

One of the most important potential of SOA is service chain to buildup larger application from distributed individual services. The OpenGIS Service Architecture standard [3] defines a service chain as "a sequence of services where, for each adjacent pair of services, occurrence of the first action is necessary for the occurrence of the next action". Service chain is required in a situation when the functionality required for a given task can not be satisfied by any existing single service, but by chaining suitable multiple existing services [23].

3.3 SERVICE CHAIN DESIGN TYPES

According to [2] [4], there are three service chain design types depending on the degree of transparency and coordination of the service chain to the user. These are:

- Transparent service chaining
- Opaque service chaining
- Translucent service chaining

3.3.1 Transparent services chain type

Transparent chaining is decentralized design types which requires high participation of user to the services in the chain. The user manages the entire workflow of services in the chain such as searching for participant services, sequence and interaction among services, and defining service chain. There is no always a specific service chain existing before the user start to discover services and defines the chain. The user must have also prior knowledge of the services inputs and outputs requirements. In addition, the availability of valid inputs to services and all intermediate process results are handled by the user. Generally, the services chaining complexities is fully transparent to the user. Because of the difficulties of the services coordination, this design type is not appropriate for complex chaining [2] [4].

3.3.2 opaque (aggregate) service chain type

In contrast to the transparent service chain, the opaque chain is centralized service chaining pattern. All participant services appear as a single aggregate service to the user. The aggregate service handles all the coordination of the individual participant services in the chain. The user is not aware of any information about the services in the service chain. Besides the advantage of simplicity to the user, it has its own drawbacks such as user loses some of the flexibility, control over parameters of the individual services [2] [4].

3.3.3 Translucent service chain type

The translucent service chain type provides a balance between the totally transparent and totally opaque services chaining types. Because, this design combines the simplicity of opaque chaining with the flexibility of control in transparent chaining type. The workflow (mediating) services act as gateways to other services. The details of service chain are transparent to the user such as the user is aware of participant services. But the control over the service chain is managed by the workflow service. The abstractly predefined service chain (a service chain without referring to a specific service implementation) is already stored on a workflow engine (an application which executes the service chain). The user requests to execute predefined services controlled by single or multiple workflow service (s). Based on the abstractly predefined services chain, the workflow service determines appropriate data sources and processing services, control sequence of execution, and assemble the final result and present to the user [2] [4].

The three service chain design types can be integrated in different ways. For example, a service chain designed based on transparent service chain type, which produce a correct result, then this chain can be exposed as a new service following either the opaque or translucent service chain type.

3.4 SERVICE CHAINING PROCEDURE

Before having one of the above mentioned service chaining types, a method is required to create the service chain. As the services are distributed application, procedural steps can be undertaken to combine those services into service chains.

- Abstract service chaining: process of identifying service chain functionality without referring to specific services implementation.

- Service discovery: process of searching services from repository of services which involves matching between the description of services and description of given task.

- Concrete service chaining: process of creating specific services chain for execution by identifying their required inputs and order of execution for each participant services based on the abstract service chaining.

- Service chain execution: process of implementation of the executable service chain which are created in concrete chaining step.

3.4.1 Abstarct service chaining

The objective of abstract chaining to identify potential services, defining control flow (service execution order), and data flow (how data is exchanged between services) with out refereing to any specific service implementation. The control flow in service chain defines the order of services execution within a service chain. There three control flow design options: sequential, parallel, and alternative.

- Sequential control flow: In sequential control flow is the simplest control flow in service chain, where services are executed one after another in predefined order.
- Parallel control flow: In parallel control allows two or more service to executed at a time in a service chain.
- Alternative control flow: In alternative control flow service execution is determined depending on a condition having multiple direction of execution.

The data flow of service chain defines how data is exchanged between services. Services have input and output parameters. In a service chain, the output of one service can be used as input for another service.

3.4.2 Service discovery

In order to start chaining of specific services, the services to be chained needs to be known. Knowing services starts from getting the description of each services. This description can be retrieved through service discovery process from services' repository. A services' repository refers to a storage where services' description are located. For example the OGC CSW is repository which supports user to discover geoservices' description.

The discovery of candidate service for the given task involves the comparison and evaluation of services' description and requirements of the given task. This can be done either using discovery application or manually by reviewing the content of the services' description. The content of services' description includes service's functional properties (i.e what a service does) and non-functional properties (i.e how a service does) [23].

- Functional properties: Includes descriptions about the functionality offered by a service and the corresponding required input(s) and output(s) to be produced.
- Non-functional properties: Include other description which can be used to evaluate the services such as QoS (Quality of Service), cost, and security issue. The non-functional properties are additional user specific requirements. Due to this, we do not take into account these non-functional properties as requirements to create service chain (out of scope of this research). We only focus on the functional properties as they are the fundamental required properties which allow to use a services' functionality.

In service discovery process, it quite common case to discover more than one service which have similar functional fitting the requirements of a given task. On the other hand, the wrong service may be discovered depending on discover technique used. For example, the WPS processing-functionalities are described by *DescribeProcess* response document. The syntax (structure of input and output) of each process is described by *DescribeProcess* response document. But, semantics (meaning of inputs and outputs) is not included. Consequently, the discovered processing-functionalities may be wrong for the given requirement. This problem is exemplified in [19] as the following. For example, Euclidian distance can be used to find the shortest distance measured in 2-dimentaional plane, or a shortest Euclidian distance in 3-dimensional space. The two

WPS processing-functionalities provide different functionality with different input requirements, although both are described by the same (syntactic) description. To overcome those problem semantic description is proposed as a solution.

In contrast, there may a case that no service found that fulfill requirement of the given task. In such condition, requirement of the given task needs to be refined into sub requirements until a set of services are discovered. In case of more than service are discovered with same functional properties while fitting requirements of given task, in order to determine which candidate service should be selected to be combined in to service chain, the following selection criteria which can have a key role to support for decision on selctting the best services:

- The first criteria can be selecting a service which needs minimum adaption for initial inputs requirement such as first initial input data type, format, the required output. In addition, the adaption required between one services to another service.
- The second criteria can be, selecting two or more services which are within the same service provider. This will have an advantage to increase performance since the message exchange speed rate can be increased.
- The third criteria can be, using the non-functional properties such as cost and response time can play key role to rank and select the best fittest services. The ranking can be done either by human user giving his relevance weight or using software application.

In case of no service is found, and then the requirement definition should be refined in to smaller requirements which should be to be achieved. Having the awareness of the above mentioned service discovery problems, we continued assuming that a set of fittest service are discovered for the given task.

3.4.3 Concrete service chaining

Concrete service chaining is the process of creating an instance of abstract chain. In this process, specific participant service instances, candidate services which are discovered and selected fitting the requirements of the given task are chained where the required inputs and sequence of execution is defined. In concrete service chaining, the chainability of these candidate service needs to be evaluated.

The chainability of two services can be determined by the compatibility of the two service description (interfaces), as interface is the point of interaction between services. For example, if there are two services, source service and target service, which are candidate service to be chained to perform complex task. The chainability of these two service can be determined by the coverage of expected inputs of target service by the output of source service in terms of number and data type, and format. Based on these conditions, we identified four chainability classification from web services chaining literatures in [6] [19].

- Exact (fully chainable): if the output of the sources service covers directly all required inputs of target service. For example, on (figure 3.1, example No-1), assume buffer, source service, requires two inputs, first input(input1) is polygon in GML format and second input (input 2) is distance, and produces one output(Bufferedpolygon) in GML format. GetArea , target service, requires one input(polygon) in GML format. Here as we can see , the output of buffer can be directly used as input to the GetArea which is indicated by green arrow line which means the two services are fully chainable.

- Partially chainable: if the outputs of the source service can cover directly one or more (not all) required inputs of target service. For example, on (figure 3.1, example No-2), assume intersection, source service, requires two inputs, first input(input1) is polygon in GML format and second input (input 2) is LineString in GML format, and produces one output(newgeometry) in GML format. Buffer, target service, requires two inputs, first input(input1) is polygon in GML format and second input (input 2) is distance, and produces one output(Bufferedpolygon) in GML format. Here as we can see , the output of intersection which is indicated by green arrow can be used as input directly for buffer, but buffer requires additional input, distance which is indicated by the red arrow to mean the input is not covered by the source service. Therefore, the two services are partially chainable.

- Adaptable fully chainable: if the outputs of source services can cover all required inputs, but adaption of one or more output of sources service or input of target service or both are required. For example, on (figure 3.1, example No-3), which is the same as (figure 3.1, example No-1), but here the source and target service supports different coordinate system(SRID) which is indicated by red arrow to mean the output can not be directly used as input to the target service, instead adaption or manipulation of this output is required so that the two services can participate still in the chain.

- Adaptable partially chainable: if the outputs of the source service can cover one or more (not all) required inputs of target service but, adaption of one or more output of sources service or input of target service or both are required. For example, on (figure 3.1, example No-4), is the same as (figure 3.1, example No-2), but here for two cases the source does not cover the requirements of the target service. The first case all input are not covered, which is indicated by red arrow, secondly, the output of source can not be directly used as input to the target, instead adaption or manipulation of this output is required.

However, to determine such classification of services chainability, there should be either a metric unit that can measure the degree of their similarity by parsing their description input and output description with some specified threshold automatically, or an expert user is required to determine their chainability by reviewing the services' description.

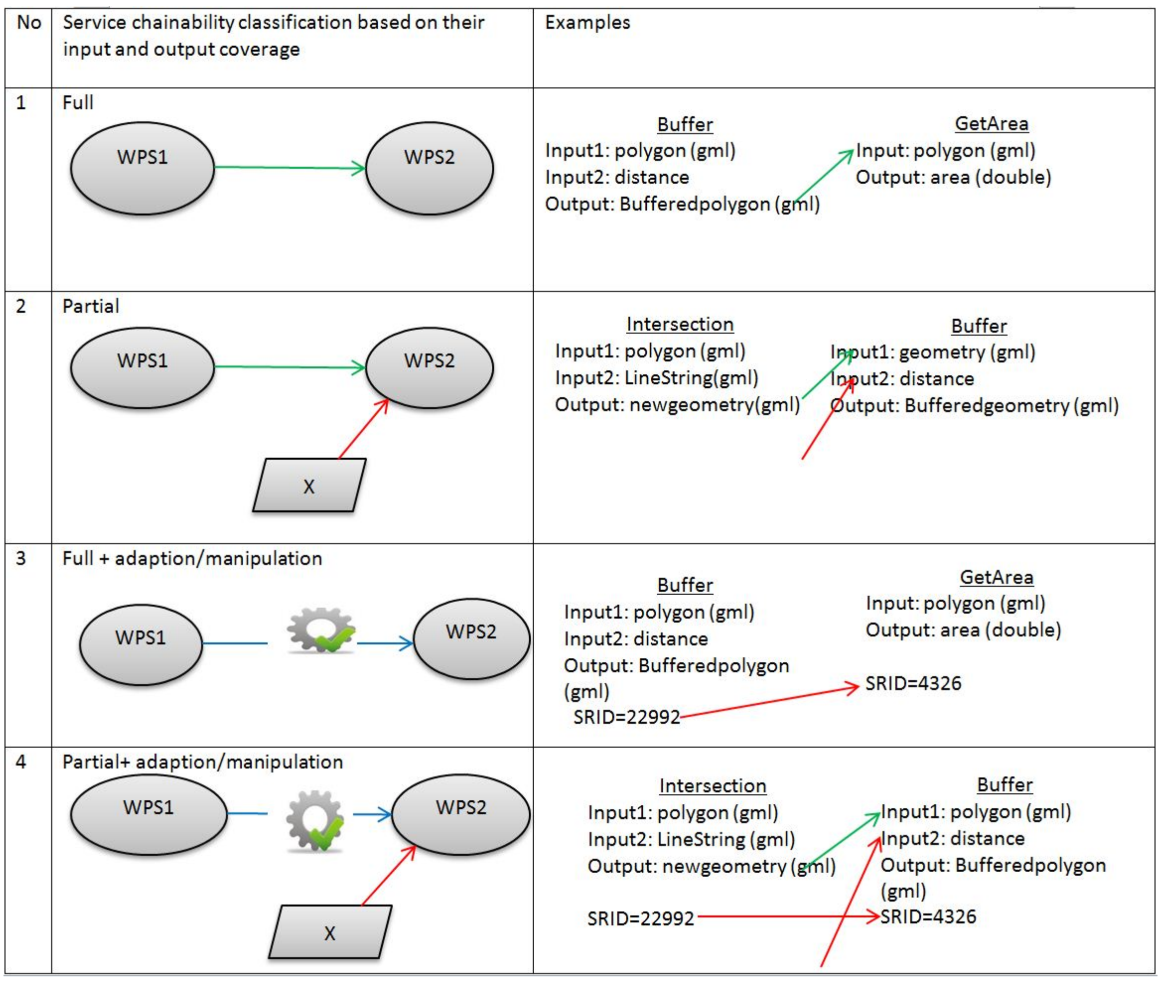
3.4.4 Service chain execution

Having an executable services chain, created in concrete chaining step which are ready for their execution, the next step is service chain execution. Service chain execution is the action of triggering each participant service to take input(s) and produce output(s), and passing the output of one service, service executed before, as input to another service, to be executed next, based on their defined sequence.

For the execution of the service chain, an implementation application that can interpret and execute each participant services is required. For geoservices, there are three implementation options: BPEL as orchestration engine, WPS for centralized service chaining, and WPS for cascade chaining.

BPEL as orchestration engine

[1] is the popular commonly known XML based language which allows the execution of complex service workflows. As we highlighted in chapter 1.1, BPEL is being used widely to orchestrate service chains mainly in Information Technology (IT) stream. BPEL relies on WSDL [10] for interface description of participant services in the service chain, and SOAP for message exchange



No	Service chainability classification based on their input and output coverage	Examples
1	Full	Buffer Input1: polygon (gml) Input2: distance Output: Bufferedpolygon (gml) GetArea Input: polygon (gml) Output: area (double)
2	Partial	Intersection Input1: polygon (gml) Input2: LineString(gml) Output: newgeometry(gml) Buffer Input1: geometry (gml) Input2: distance Output: Bufferedgeometry (gml)
3	Full + adaption/manipulation	Buffer Input1: polygon (gml) Input2: distance Output: Bufferedpolygon (gml) SRID=22992 GetArea Input: polygon (gml) Output: area (double) SRID=4326
4	Partial+ adaption/manipulation	Intersection Input1: polygon (gml) Input2: LineString (gml) Output: newgeometry (gml) SRID=22992 Buffer Input1: polygon (gml) Input2: distance Output: Bufferedpolygon (gml) SRID=4326

Figure 3.1: Services chainability classification

mechanisms. WSDL serves as a link between the BPEL and participant services [26]. BPEL can be also applied to orchestrate geoservices chains by describing the interaction among participant services. WPS also supports both SOAP and WSDL. Therefore the incorporation of WPS to BPEL engine to be orchestrated can offer more sophisticated service chaining capabilities [25]. However, using BPEL requires each participant geoservices to be described in WSDL. Unfortunately, most OGC geoservices are not described using BPEL [30]. Therefore, to use BPEL, a WSDL needs to be generated for each services.

For the case of WPS, the specification provides an option to link WSDL as apart of *DescribeProcess* response to be providers if they are willing. In such condition, BPEL can be applied as orchestration engine. However, as WSDL is not always provided by providers as it an additional task for them. Therefore, consumers should generate WSDL document for each required processing-functionalities to be orchestrated. This is going to be tedious task to generate such document individually.

WPS for centralized service chaining

The second possible approach is chaining set of geoservices, including processing-functionalities, and the chaining is encapsulated with WPS interface, serving as chaining engine to execute a service chain [25] by calling one after another based on their sequence. This approach is an important approach to create aggregate services out of existing services. The opaque or aggregate service design type can be achieved with this approach. Our chaining approach will be inline with this approach.

WPS to cascade chaining

The other third option is using simple cascading service chain via WPS *Execute* request which can involve two or more number of processing-functionalities. The chain can be encoded using KVP (Key-Value-Pair) where a call to another processing-functionalities is encoded as complex data type and send using web browser. However, this approach is not valid for more than two or three processing-functionalities in the chain. Because, as the number of participant processing-functionalities increase, the length of encoding increases and the web browser may not support due to its limited number of characters to handle.

3.5 SUMMARY

In this chapter we have identified different approaches of chaining patterns , procedural steps. In addition to the procedures and approaches to create services chaining, the aim of the this chapter was to identify the requirements for chaining two or more services. In this analysis we identified that the description of a service is required before using a service for any application. The interaction of a service with another service is also determined with this description. As a result, two or more services are chainable when the output of the first service can cover at least one input expected by the next service. This requirements is used as input to design chaining method in chapter 4.

Chapter 4
Method for chaining of WPS processing-functionalities

4.1 INTRODUCTION

The objective of this chapter is to design a generic chaining schema which defines all possible required contents conceptually that can serve as a template for chaining of any two or more number of processing-functionalities in any GI application domain. In addition, after design of this generic chaining schema is completed, architecture for chaining of processing-functionalities using this generic chaining schema is provided. The chapter starts with defining the role of the generic chaining schema in section 4.2, service chaining requirements analysis in section 4.2.1, *DescribeProcess* response parameters analysis in section 4.2.2, generic chaining schema design in section 4.2.3, and architecture of chaining processing-functionalities in section 4.3

4.2 GENERIC CHAINING SCHEMA

To combine two or more number of processing-functionalities into service chain, first a generic mechanism is needed to be designed that can be used for any chaining requirement independent of any GI application. To design such generic mechanism, it is necessary to capture the essential functional properties and organize in a way to enable two or more processing-functionalities interact with each other in a chain. The interaction includes, the data flow between processing-functionalities, i.e., using output of one processing-functionality as input to another processing-functionality. In addition, control flow of each processing-functionalities within a chain i.e. in what order that each processing-functionalities should be executed is also another requirement to be captured. To capture such requirements, a schema is required. As we discussed in section 2.3, a schema is a mechanism used to capture and organize relevant information, leaving unnecessary details, out of large amount of information helping to interpret and use easily. Although, there are different schema such as relational database schema and XML schema definition (XSD) language, our requirement can be satisfied by XSD, as it is the basis for all web service development.

The target users of our designed chaining schema are expert users who are going to chain distributed WPS processing-functionalities, in case of individual processing-functionalities are insufficient to perform a given geospatail task.

Before starting designing chaining schema, the necessary functionalities which should be considered in designing the chaining schema needs to be identified. In order to identify the functionalities, we used two requirement analysis as input to design the chaining schema as shown on figure 4.1.

1. Service chaining requirements analysis : which is already done in chapter 3, is analysis from web services and geoservices domain in order to identify the basic requirements that needs to be considered in the design of chaining schema.

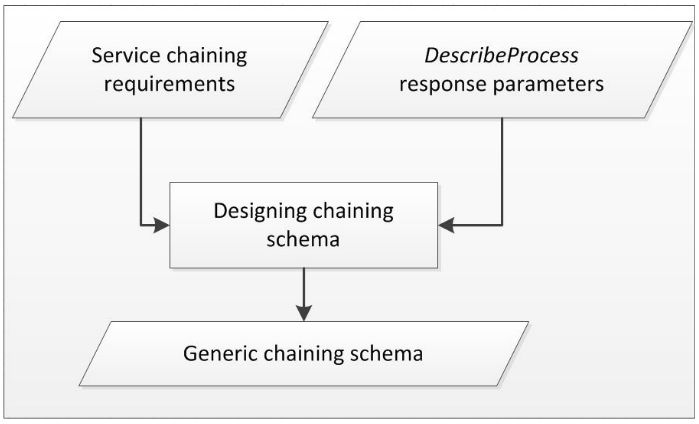

Service chaining requirements
DescribeProcess response parameters
Designing chaining schema
Generic chaining schema

Figure 4.1: Approach to generic chaining schema

2. *DescribeProcess* response parameters analysis: is the *DescribeProcess* response analysis from WPS specification as well as from real-world processing-functionalities' *DescribeProcess* responses content. From this analysis we identified the required parameters for a processing-functionality to be considered in designing the chaining schema.

3. Design chaining schema using step 1 and step 2: using the two analysis as input, we designed the chaining schema using XML schema definition (XSD) language. The role of this chaining schema is to represent building block of all possible processing-functionalities chaining. We use the XSD schema to model the required component for processing-functionalities chaining and their inter-relationships. The relationship in the schema indicates the structure of the processing-functionalities to be described in chain. We captured all required parameters from the *DescribeProcess* response parameters and model as schema elements containing subelements. So that the required information for chaining of processing-functionalities can be captured and described using these elements.

4.2.1 Service chaining requirements analysis

In analyzing web service and geoservices chaining requirement in chapter 3, we knew that the description of any service is required before using it for any application. The interaction of a service with another service is also determined with this description. The description content is comprised of functional and non-functional properties of services. We chose to consider only the functional properties which includes name of functionality, its input requirement, and output to be produced as they are the fundamental requirements to use a service. Moreover, two services can be chained if and only if the output of the first service can cover at least one input expected by the next service.

4.2.2 *DescribeProcess* response parameters analysis

For our case, the description requirement is fulfilled by the *DescribeProcess* response content containing a list of processing-functionalities with full description including input and output parameters. Due to this, we chose to analyze this content to capture the relevant parameters that each participant processing-functionality should bring to be chained using chaining schema.

We started first by analyzing the *DescribeProcess* response content of the WPS specification provided on table 2.4. From this analysis, we have identified the mandatory and optional parameters, the number of inputs that can be accepted, the number of outputs that can be produced, the capability that inputs can be provided either by value or by reference, and the capability of returning outputs either as reference, raw data, or encapsulated (wrapped with XML document) of any processing-functionalities, in general.

In addition to the specification, we have inspected a number of real-world *DescribeProcess* response document instances from randomly selected WPS processing-functionalities providers. The samples are intersection and buffer processing-functionalities (see appendix 6.3) from ZOO [1] WPS and Deegree [2] WPS using their base URL found on "Open Registry" catalogue [3].

As we noticed, the major difference between the response document from different WPS service providers differs in providing and not providing parameter values which are optional in the specification such as WSDL, storage, and status response services. Among the two providers, the Zoo WPS only provides both storage and status services for the processing-functionalities we use for our analysis. This indicates that using reference as input transfer mechanism from one participant processing-functionality to another is not granted. From this analysis, the selected parameters which are most important to create service chaining are: processing-functionality's identifer, storeSupported, statuSupported, Datainputs, input identifier, ProcessOutputs, and output identifer. According to the specification their description and role is stated in table 2.4.

4.2.3 Generic chaining schema design

Based on the selected parameters from the *DescribeProcess* response content and additional parameters which were considered to be included, the generic chaining schema is designed as shown on figure 4.2. As we can see on the figure, the chaining schema consists of all necessary information for execution of any processing-functionality. The design considerations and role of each elements in the chaining schema description is as follows.

- ProcessingFunctionality: stands for a single atomic WPS processing-functionality.
- In generic chaining schema, two or more number of processing-functionalities can be involved.
- Each participant processing-functionality must have a base URL (Uniform Resource Locator). This base URL is a pointer to where the participant processing-functionality resides on the world and can be accessed for execution.
- Each processing-functionality must have a name (identifer), the same as the identifer specified in the *DescribeProcess* response document.
- In addition to the identifer (name), each processing-functionality must have a unique position identifier within a chain used to identify one processing-functionality from another. This identifer is specially important when the same processing-functionality is used multiple times with different input values in the same chaining instance. For example, assume an intersection from Zoo WPS is used to execute the intersection of two polygons, then followed by buffer to create a polygon around the result of the intersection with a certain distance. Again, if we want to intersect the result of buffer with another polygon, then we may use the same intersection to execute this task, with different position in chain identifer within the same chaining instance.

[1]http://zoo-project.org/
[2]http://flexigeoweb.lat-lon.de/deegree-wps-demo
[3]http://openregistry.info/registry/Find_client

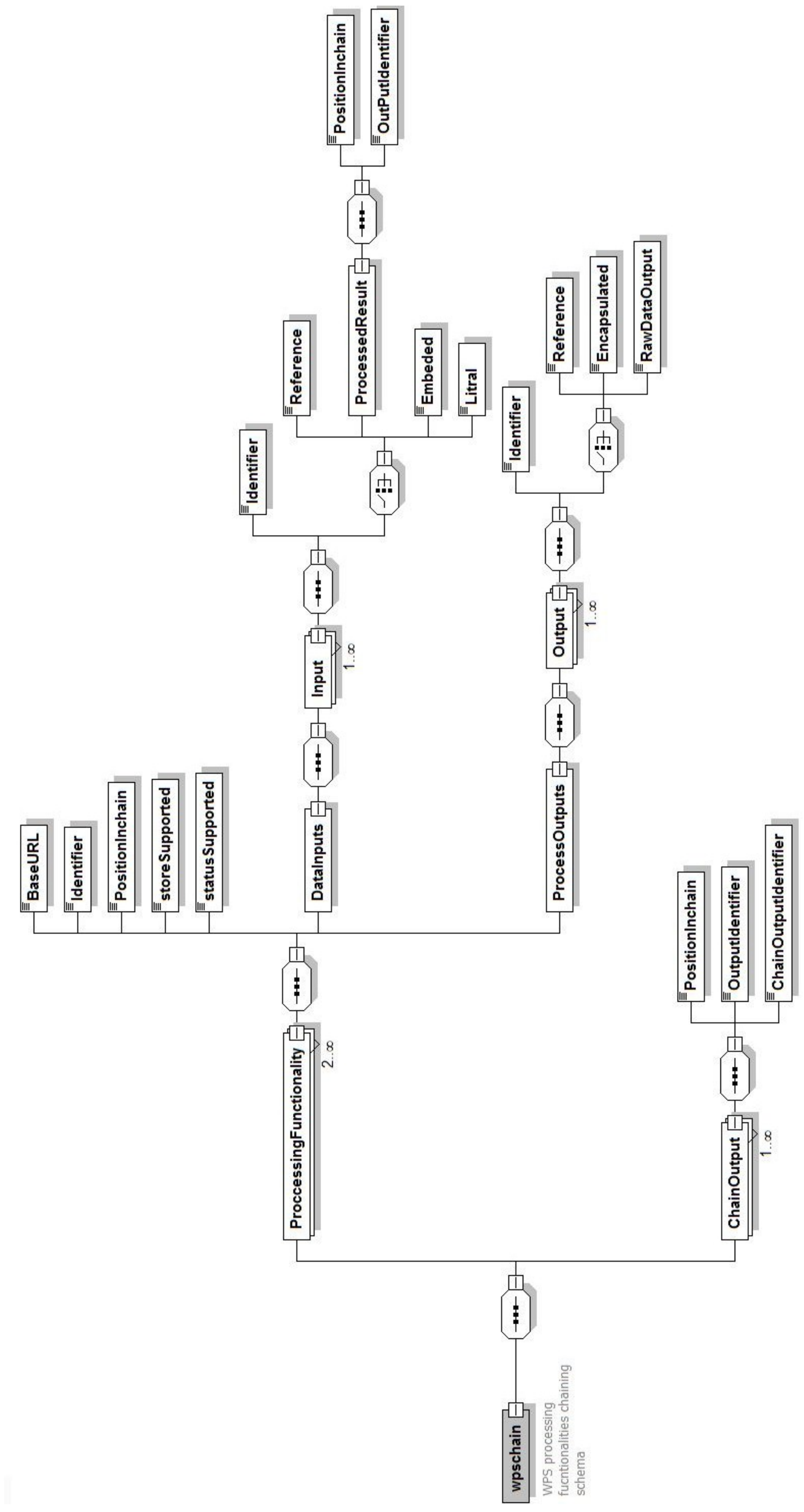


Figure 4.2: Generic chaining schema

- Each processing-functionality should have information on the store supported and status supported as optional parameters. The value may be either true or false. If storage supported is true, then output return with the chain can be instructed by reference. And if the status value is true, then the progress of completed task for the case of long-running task execution can be retrieved.

- Each processing-functionality in the chain can take more than one inputs as WPS specification allows a processing-functionality to take any number of inputs. Each input must have a unique identifer, same identifer which is specified in the *DescribeProcess* response content. As the WPS specification offers flexibility to choose on different inputs types, inputs to each processing-functionality can be either reference input, or processed result, or embedded input, or literal input. These input types are captured by the chaining schema as shown on figure 4.3.

  - Referenced (URL) input: is input value provided as URL which is a link to the actual data such as WFS *GetFeature* request, or URL output which is returned by a processing-functionality and passed as input to another processing-functionality within the chain.
  - Processed result input: is output of previously executed processing-functionality within a chain. This output can be passed as input for another participant processing-functionalities within the chain. In such case, the output identifer and position in chain identifer of the previous processing-functionality to be executed must be passed as input to the next processing-functionality.
  - Embedded input: is the input which is provided directly by actual value to be passed with the execute request parameters. This input type is appropriate for small sized data coordinate pairs, but not for large data, specially when execute request is sent using the HTTP GET KVP method, the browser can not handle the entire data, as we proved. Therefore, for larger data, providing data by reference is more applicable than by value.
  - Literal input: literal inputs are simple textual values such as buffer distance.

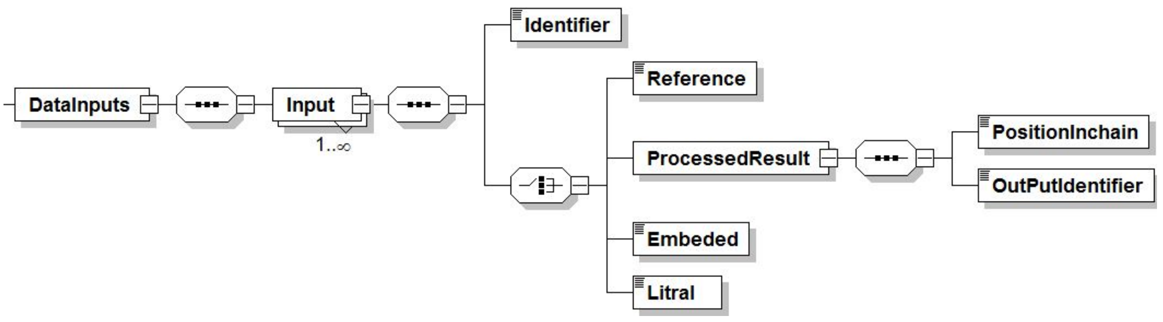


Figure 4.3: Input types in generic chaining schema

- Each processing-functionality in the chain can produce more than one outputs, as WPS specification allows a processing-functionality to produce any number of outputs. In a similar case with input value types, the WPS specification also offer flexibility to choose on three output response options: raw data output, referenced (URL) output, or encapsulated output (XML wrapped output). These output types are captured by the chaining schema as shown on figure 4.4.

- Referenced (URL) output: This output response type is dependent on the storage service supported by the participant processing-functionalities. This information can be known by checking the boolean value *statusSupported* parameter in *DescribeProcess* response content. If true, then the processing-functionality which support the storage services can be requested to store the output and return only a reference (URL) to access the output.
- Encapsulated output: This option is the normal response type after each WPS precessing-functionalities are executed unless another response type is requested. For example, such as GML output returned encapsulated by XML execute response document.
- Raw data output: In this output response type, the data is delivered back as raw data without encoded in a message structure format such as in XML document. For instance, if a Geotiff image is requested, the image is delivered in binary format and not wrapped by XML document. However, this request is supported for a single output only which can be delivered at a time since the output is not encoded in any message structure format.

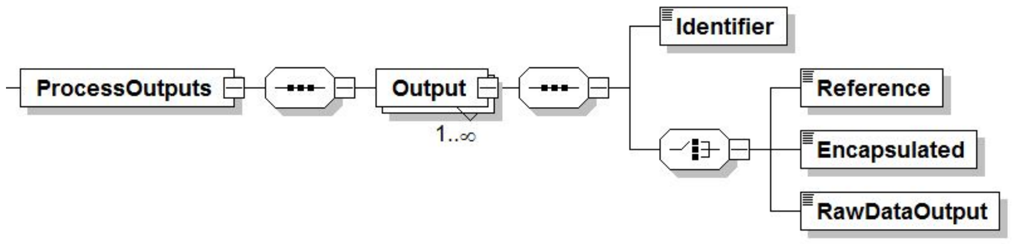


Figure 4.4: Output types in generic chaining schema

- The final elements of the chaining schema is chain output which represents final output of a chain. In our chaining schema, the output of the chain is equal to the output of the last processing-functionality to be executed. The chain output element plays role to select the required output only in case of the last executed processing-functionality has more than one output. In such case, this element allows to identify that required output identifier from *DescribeProcess* response content as well as position in chain identifier of that processing-functionality and assign to this chaining output element. In addition, the chain output can have its own additional identifer which can serve as the overall chaining output identifer as we can see on figure 4.5.

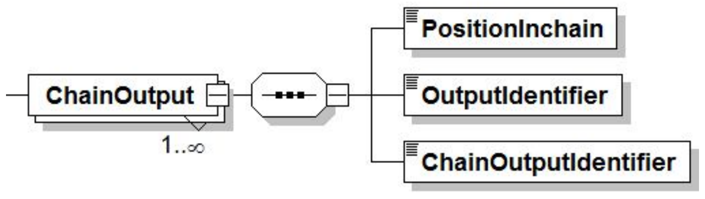


Figure 4.5: Final output in generic chaining schema

Now, the development of generic chaining schema is completed. This generic chaining schema captured all necessary functional requirements for combining a set of any processing-functionalities into service chain to perform complex geoprocessing task independent of GI application domain. As we mentioned in section 4.1, in the remaining section of this chapter, we focused to provide an architecture of chaining one or more number of processing-functionalities using this generic chaining schema, as it is ready to be used.

4.3 ARCHITECTURE OF CHAINING PROCESSING-FUNCTIONALITIES

Using the designed chaining schema, any two or more number of executable chained processing-functionalities instances can be derived. The architecture for chaining of processing-functionalities is provided as shown on figure 4.6 which consists of processing steps (e.g mapping rules implementation), outputs (e.g mapping rules), inputs (e.g *DescribeProcess* responses), and methods (e.g generic chaining schema) which are used to drive such executable chain instance. Brief description on each elements of the architecture is provided below. Then the main elements are also discussed in the following sections.

- Expert user: the involvement of an expert user is required in order to evaluate the validity of the inputs, the chainability of processing-functionalities, final output of the chain, and to perform the chaining process using the chaining schema with interpretation of their *DescribeProcess* responses content.
- Chain requirement analysis: is the process of identifying the required processing- functionalities to be discovered.
  - Chain requirement descriptions: is the output of the chain requirements analysis which is a list of required processing-functionalities including their order of execution, initial inputs, and required output for a given task. These information are the basis to start discovery and chaining process.
- *GetCapabilities* request: a process required to retrieve a list of existing processing- functionalities within a single WPS server using request parameters provided in table 2.1. This request is useful to get the identifier (name) of each processing-functionality to send *DescribeProcess* request.
  - List of processing-functionalities: one or more number of processing-functionalities which are returned from *GetCapabilities* request.
- *DescribeProcess* request: a process required to retrieve the *DescribeProcess* response of each processing-functionalities using request parameters provided in table 2.3.
  - *DescribeProcess* responses : for each processing-functionality, the selected parameters values needs to be populated from their *DescribeProcess* response to the corresponding elements of chaining schema.
- Generic chaining schema : the chaining schema is the place where a set of participant processing-functionalities are chained. This process is done by populating the elements of the chaining schema with the selected parameters values from *DescribeProcess* response content of each processing-functionality.
- Mapping *DescribeProcess* responses to chaining schema: is a process required to select and populate the chaining schema with the *DescribeProcess* response parameters values.

  - Mapping rules: rules about the relationship between the selected *DescribeProcess* response values and the corresponding elements of chaining schema

- Mapping rules implementation: is a process of interpreting and executing the mapping rules. So that the actual selected values can be get written to the chaining schema.

  - Chain instance: is the output of mapping rules implementation which is a single XML document created based on chaining schema which contains a set of ordered participant processing-functionalities with all required information for their execution.

- Chaining engine: is required to interpret and execute content of chain instance.

  - Final result: is the complete chain output after the execution of the chain instance by the chaining engine .

4.3.1 Expert user

The application of the chaining schema needs the involvement of an expert user, the target user, as we motioned in section 4.1. We assumed the correctness of over all chain output is determined by this expert user. The correctness can be determined based on three criteria which are listed in [2] which needs interpreting *DescribeProcess* response content of each processing-functionalities.

- Appropriateness of initial input data: for each participant processing-functionality, validity of the required inputs must checked. For example, the accepted geometry type and format, polygon, or point, and the format in GML, or in shape file must be checked and the required input must be provided.

- Impact of processing-functionality on the input data: how the output of individual processing-functionality change the chaining result.

- Sequence of processing-functionality: The order of processing-functionalities to be executed must be determined based on the required result and the output of the processing-functionalities. For example, is intersection of the road occur first and then to buffer from the result of the intersection, is the output of the intersection a valid input for buffer processing-functionalities. Is the last processing-functionalities output can return the required output.

4.3.2 Chain requirement analysis

As we discussed in service chain concept, section 3.2, chaining of processing-functionalities is required in a situation when the functionality required for a given geoprocessing task can not be satisfied by any existing single processing-functionality, but by chaining suitable multiple existing processing-functionalities. In order to chain a set of existing processing-functionalities, the chain requirement for a given geoprocessing task needs to be analyzed first. Through chaining requirement analysis the required processing-functionalities to be discovered, initial inputs, required output, and order of processing-functionalities to be executed to get the final required output of the given geoprocessing task can be identified. So that using these chain requirement list of description, the existing appropriate processing-functionalities can be discovered which satisfies the chain description.

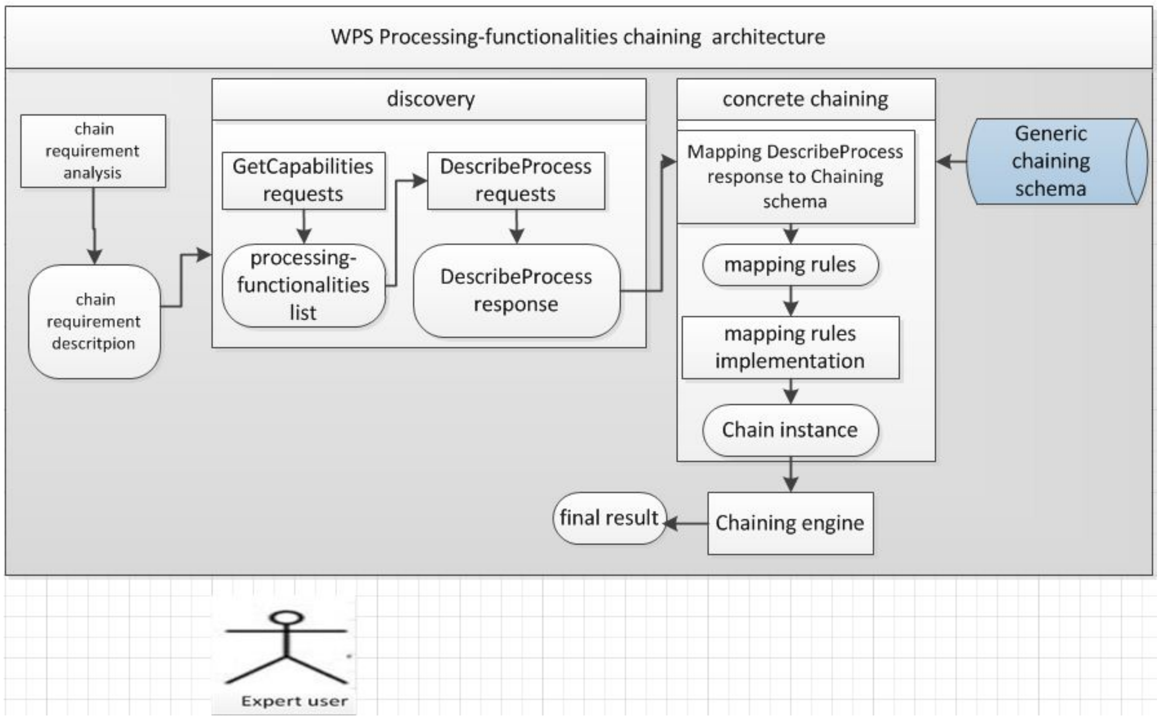

WPS Processing-functionalities chaining architecture
chain requirement analysis
chain requirement descritpion
discovery
GetCapabilities requests
processing-functionalities list
DescribeProcess requests
DescribeProcess response
concrete chaining
Mapping DescribeProcess response to Chaining schema
mapping rules
mapping rules implementation
Chain instance
Generic chaining schema
Chaining engine
final result
Expert user

Figure 4.6: WPS processing-functionalities chaining architecture

4.3.3 *DescribeProcess* responses

Using list of chain requirement description, discovering and selecting relevant chainable processing-functionalities from existing processing-functionalities which can satisfy chaining requirement description is an important step. To discover a set of existing processing-functionalities, their base URL or entry point of WPS server is required. In real-world scenario, such base URL can be retrieved from catalogue service. For our case, we assume that the expert user knows such base URLs of the existing WPS servers.

As we have discussed in services chaining requirements analysis in section 3.4.3, the chainability of two processing-functionalities can be determined with the output coverage of the former processing-functionality as input to later processing- functionality. In order to take those determination, the *DescribeProcess* response document is required. For our case, this comparison and evaluation process is the responsibility of the expert user by discovering the *DescribeProcess* response document of each processing-functionalities and selecting the relevant chainable processing- functionalities. Relevant chainable processing-functionalities represent processing-functionalities that minimize the amount of required adaptation among processing-functionalities while best fitting for requirements of given task.

4.3.4 Generic chaining schema

The designed chaining schema itself is the place where a set of participant processing-functionalities are chained. This process is done by populating the elements of the chaining schema with the selected parameters values from *DescribeProcess* response document of each processing -functionality.

4.3.5 Mapping *DescribeProcess* responses to chaining schema

Having, all required information such as *DescribeProcess* responses for each processing- functionalities, knowing their order within the chain, and valid initial inputs, the selected contents in *DescribeProcess* response of each processing-functionality needs to be selected and written to the corresponding elements in chaining schema based on specified order. In order to facilitate the selection of required parameter values from *DescribeProcess* response and populate to chaining schema, a schema mapping needs to be created between *DescribeProcess* response (source schema) and generic chaining schema (target schema). The schema mapping is needed to identify and establish the relationship between the required parameter values from *DescribeProcess* response and the chaining schema to provide the mapping rules. The mapping rules are information about which value from *DescribeProcess* response to be assigned to which elements of the chaining schema.

Through this process , a concrete processing-functionalities chain with the real-world processing-functionalities *DescribeProcess* responses is being created. The output of one processing-functionalities to be passed as input to the next processing-functionality is also determined here. As our chaining schema has no evaluation mechanism to check whether the participant processing-functionalities are chainable or not, the expert user has to evaluate the chainability of processing-functionalities by interpreting the *DescribeProcess* responses of each pair of processing-functionalities.

As we discussed in service chainability classification scenario in section 3.4.3, the expert user may found four cases of chainability classification. In case of partially chainable, two processing-functionalities can be chained because the extra required inputs can be provided by the expert user. In case of adaptable fully chainable and adaptable partially chainable, since additional intermediate processing-functionality is required, the expert user has to refine the discovery and chain requirement analysis steps in order to discover another either the intermediate processing-functionality or another pair of fully or partial chainable processing-functionalities. By this process, we assume, that all participant processing-functionalities are fully chainable.

4.3.6 Mapping rules implementation

The mapping rules produced during mapping *DescribeProcess* responses to chaining schema needs to be interpreted by a schema mapping languages for their actual implementation, i.e for populating the selected values to generic chaining schema. Schema mapping languages are languages used to encode the schema mapping rules. The mapping rules can be expressed in different languages. XSLT (Extensible Stylesheet Language Transformation), an XML based language, is one of the potential language to achieve such task [16]. The XSLT has a number of functions such as filtering elements, selection to read and extract values, condition for restrictions, and loop over elements and others functions.

4.3.7 Chaining engine

The resulted chaining instance using the previous steps is a single XML document containing list of processing-functionalities with their required parameters value such as base URL, identifer (name), inputs identifer, input values, and output identifers. This XML document needs to be interpreted and executed to get the required result. As our chaining schema is new, there exist no mechanism that can execute such XML document. For this, we have designed a chaining engine which is dedicated to execute any chaining instance which is created based on the chaining schema. The chaining engine is an executable script which should be configured based on the chaining schema elements in order to handle all possible chaining instances. The chaining engine should execute chain instance by extracting the required parameters, their corresponding values to send an execute request per processing-functionalities using the corresponding base URL.

In order to formulate the request parameters for sending a valid execute request, the following constant values are required.

- "?Request=Execute&Service=WPS&Version=1.0.0&": This constant values is always required for execute request of any processing-functionality.
- "=Reference@xlink:href=": this constant values are required when the input values are provided in referenced URL input such as GetFeature request to WFS.
- "@uom=": This constant value is required if the input value is literal.
- "storeExecuteResponse=true": this constant value is required if the storage support needs to be requested.
- "=@asReference=true": this constant values is required when the output return is requested as URL instead of the actual data
- "ResponseDocument=": is required if output return is requested is by URL or "=@asReference=true" is requested.
- "RawDataOutput=true" : is required when the output return is requested in raw data format.

In the activity diagram, these constant parameter values are also indicated on activity where they are required. For example, on figure 4.8, when input parameters are extracted, the input value type needs to be checked such as if the input value is URL (referenced input), then the concatenation of inputs parameters should be input identifer followed by the the reference constant value, "=Reference@xlink:href=".

To handle any chain instances, the chaining engine is designed to capture and interpret all possible elements of the chaining schema. To achieve this objective, we proposed the activity diagram as shown on figure 4.7, the main activity diagram, which shows the procedure to be followed for the implementation of the chaining engine.

In order to simplify the complexity of the diagram, the diagram do not contain each detail actions of the implantation, instead the main activities are highlighted. For example, if we take the activity, which says "read chaining instance content", we did not show each action how to read the chaining instance which is an XML document. Because it will be different for different programming script implementation. In addition, the activity diagram is designed for sending execution request using HTTP GET method encoded in KVP (Key-Value-Pair) for the case of simplicity, not using HTTP POST method.

The activity diagram on figure 4.7 shows, for any chaining instance , per each processing-functionality, how the chaining engine extracts the parameters and their corresponding values (such as server address where the actual processing functionality can be executed, identifier of the processing functionality) needs to be extracted.

On this activity diagram, which is labeled with letter "A" refers to figure 4.8, and "B" refers to figure 4.9.

The activity diagram on figure 4.8 shows how the input identifers and input values are interpreted and extracted for each processing-functionality. In this activity, input value type should be checked. Because, the input value can be either reference (URL), or the output of executed participant processing-functionality, or literal input value to send a valid execute request. In case of more than one input for a processing-functionality, the inputs parameters should be separated by semicolon.

The activity diagram on figure 4.9 shows that how the chaining engine interprets and extracts the output identifiers in order to specify the processed output return types and how the output of one processing-functionality can be passed to the next processing-functionality. If no return output type is specified, output return form will be the default, which is encapsulated output.

After all, the required parameters and corresponding value of each processing-functionality are extracted and concatenated, the request should be send to the processing-functionality to be executed. After the processing functionality is executed, the chaining engine should receive the output either to pass for the next processing functionality to be executed or to return to the user (in case of last executed processing-functionality).

4.4 SUMMARY

In this chapter, the generic chaining schema to chaining WPS processing-functionalities was designed. Once, having this generic chaining schema, which do not required to be designed again per chains requirement, any expert user can use it for any processing-functionalities chaining requirements. The procedural steps for chaining processing-functionalities using this generic chaining schema is also provided.

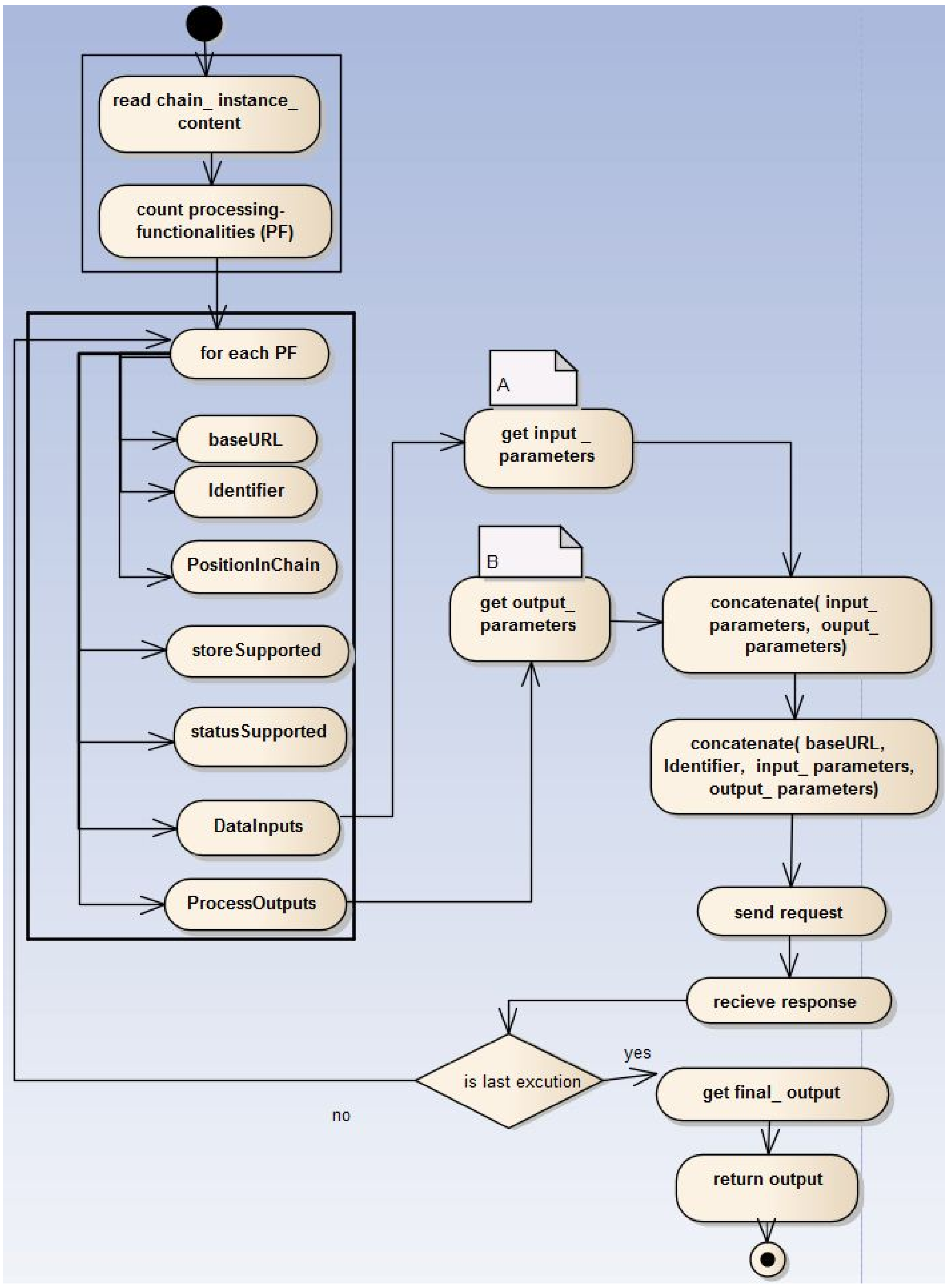


Figure 4.7: Activity diagram for execution of chaining instance by chaining engine

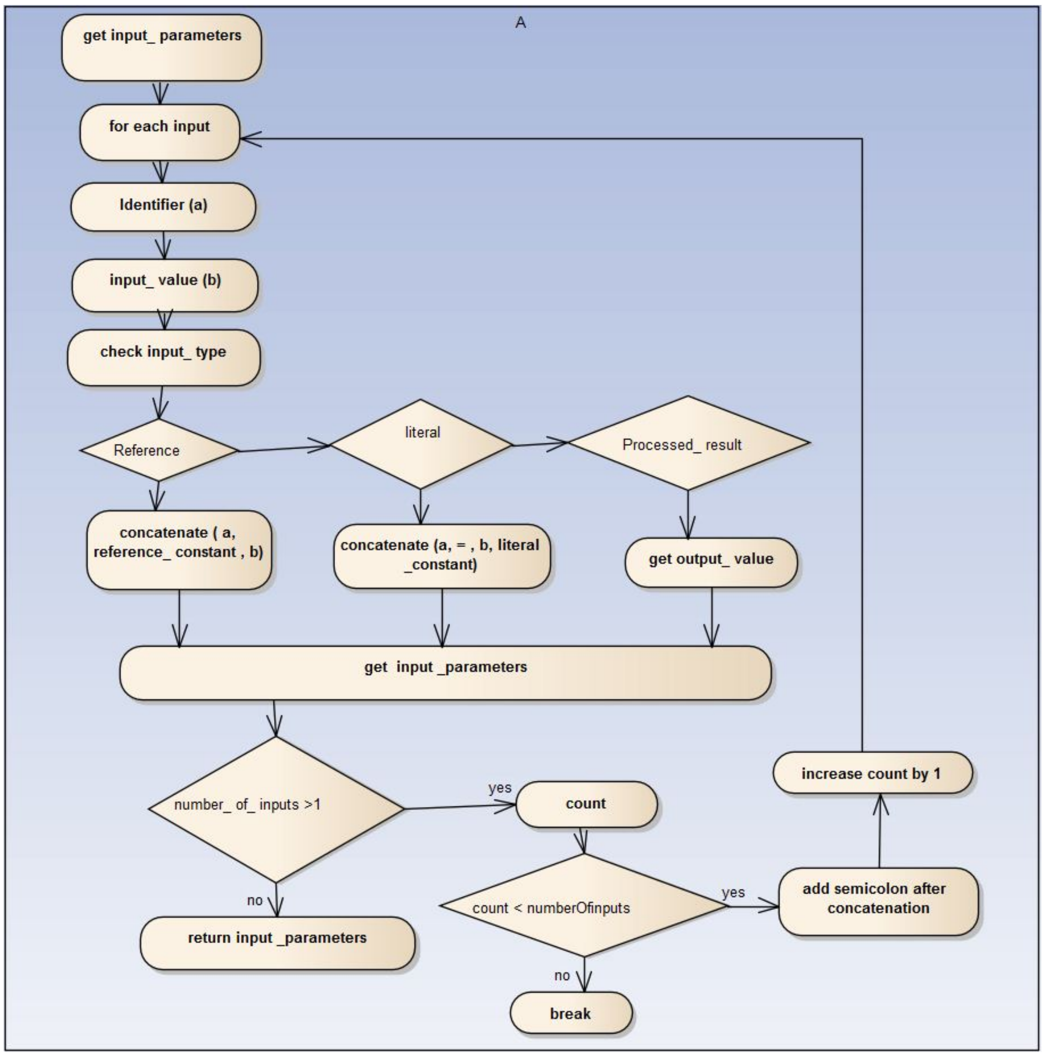


Figure 4.8: Activity diagram for input parameters extraction process by chaining engine

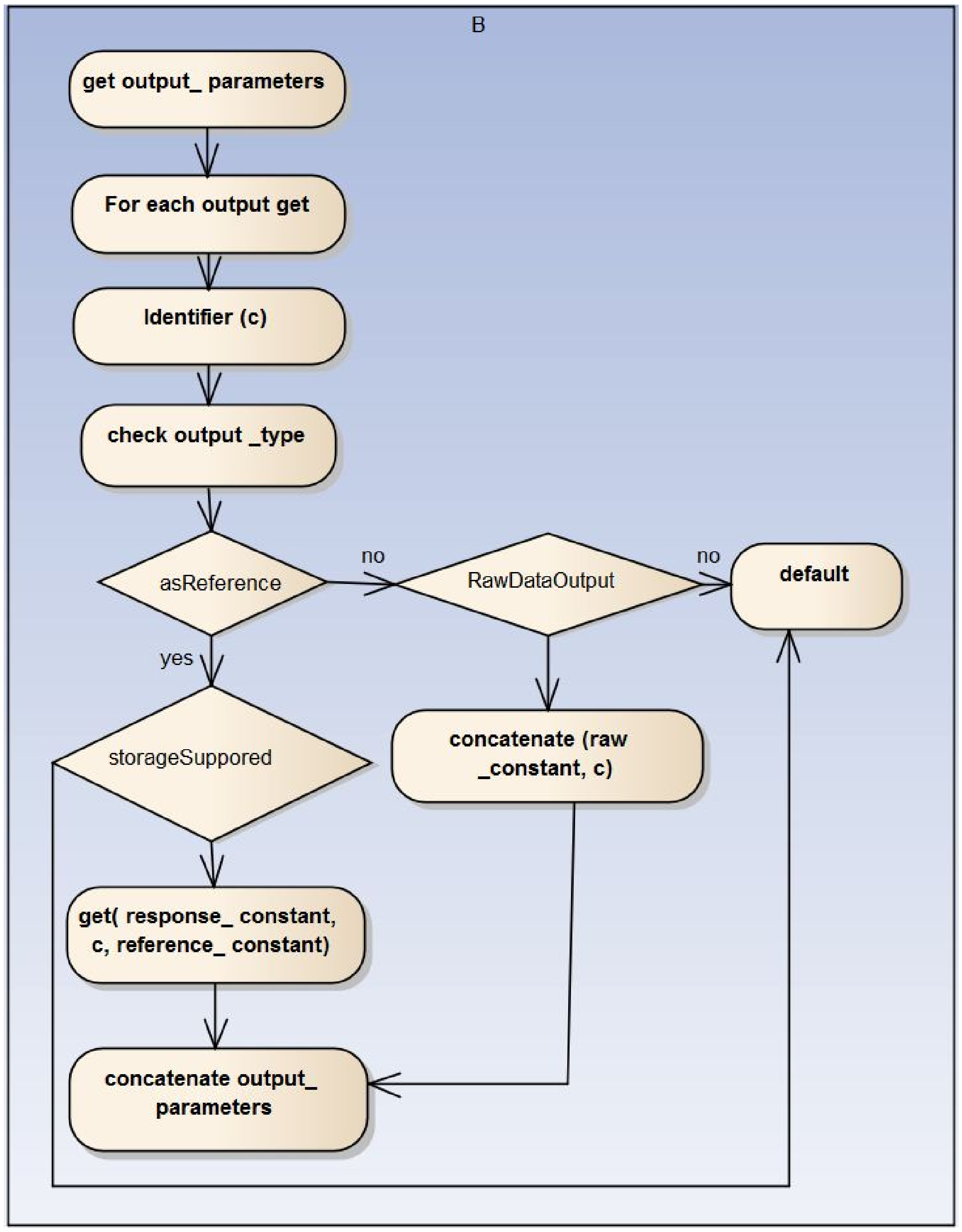


Figure 4.9: Activity diagram for output parameters extraction process by chaining engine

Chapter 5
Prototype implementation and testing

5.1 INTRODUCTION

In this chapter we completed the development of chaining method process by implementing and testing the design with application scenario as proof-of-concept. The designed chaining schema can be applied in wide application scenario independent of a specific GI application domain. We chose a real-world scenario to demonstrate how the concept works. In the prototype some number of processing-functionalities to perform the required task are involved. Using the designed chaining schema, the chaining of two processing-functionalities is defined. The chapter started with application scenario definition in section 5.2, chain requirement analysis in section 5.3, discovery of *DescribeProcess* response in section 5.4, mapping *DescribeProcess* responses to chaining schema in section 5.4.1, mapping rules implementation in section 5.4.2, chaining schema implementation in section 5.4.3, and chaining engine implementation in section 5.4.4.

5.2 APPLICATION SCENARIO DEFINITION

Searching for a land parcel is one of day-to-day activity in GI application domain. The search for land parcel may be required for different decision making purpose such as finding neighbourhood of a parcel for maintenance purpose which needs the integration of two datasets to get the required answer. For our demonstration, suppose a road in Enschede city is required to be expanded by 25 meter width. Before the expansion takes place, neighbourhood parcels which will be affected needs to be known. Assume, we are working on behalf of Enschede municipality to get parcels which will be affected within this road expansion. The given initial input datasets are Enschede parcels and road spatial data in shape file.

5.3 CHAIN REQUIREMENT ANALYSIS

To start the chaining process using the designed chaining schema, first we identified the required processing-functionalities to be involved, and their order of execution, how the initial input data should be provided, and which input for which processing-functionality, and the required output from each processing-functionality were decided. Through this process, buffer to execute the road expansion by 25 meter as first processing-functionality, and intersection as second processing-functionality to execute the intersection of parcels with the output of buffer in order to get those possible affected parcels to get intersected are identified. The chain requirement description of the given scenario is shown in figure 5.1 and is also summarized as:

- Initial input data: road and parcels datasets, both input data can be accessed via WFS *GetFeature* request in GML data format, as the the two datasets are published as WFS.

- Processing-functionalities execution order: first, buffer to create a polygon around the road. The second, intersection which takes buffered polygon as one input, and parcels as the second input.

- The final required output is parcels which are intersected with the polygon created around the road.

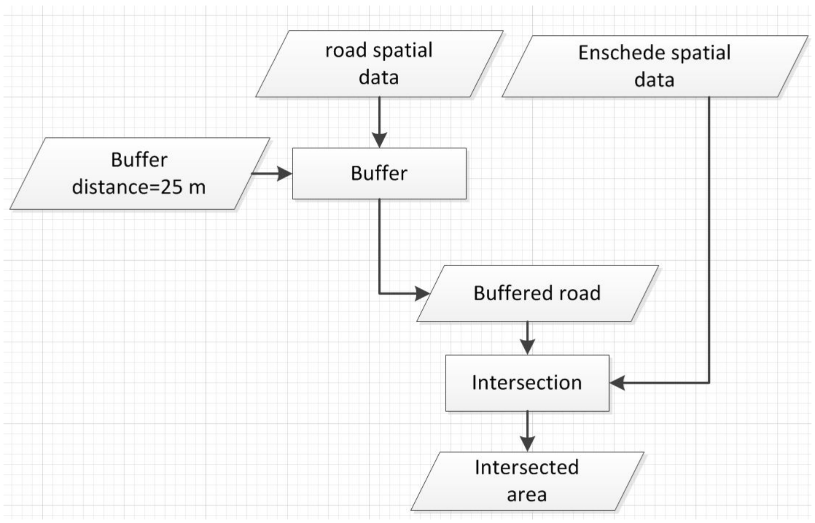


Figure 5.1: Chain requirement analysis to search for parcels

5.4 DISCOVERY OF *DESCRIBEPROCESS* RESPONSE

In order to find the existing processing-functionalities, the entry point or base URL of exiting WPS server is required. Such base URL of exiting WPS implementation can be retrieved from catalogue services. So that using this base URL, a list of available processing-functionalities can be accessed via *GetCapabilities* request using the request parameters provided on table 2.1. For example Listing 5.1 shows the *GetCapabilities* request sent to specific WPS server. The response from this request which includes list of available processing-functionalities is provided in appendix Listing 1. From this response the identifer (name) of processing-functionalities, *BufferPy* and *IntersectionPy* are identified which can satisfy the given chain requirement description. To get their full description and evaluate further, we sent the *DescribeProcess* requests using the request parameter provided on table 2.3. Listing 5.2 and Listing 5.3 shows the *DescribeProcess* requests for *BufferPy* and *IntersectionPy*, respectively. The *DescribeProcess* response of *BufferPy* and *IntersectionPy* is provided in appendix, Listing 2 and Listing 3, respectively.

Listing 5.1: *GetCapabilities* request for Zoo WPS

```
http://130.89.209.216/cgi-bin/zoo_loader.cgi?REQUEST=GetCapabilities&SERVICE=WPS&
VERSION=1.0.0
```

Listing 5.2: *DescribeProcess* request for *BufferPy*

```
http://130.89.209.216/cgi-bin/zoo_loader.cgi?REQUEST=DescribeProcess&SERVICE=WPS&
VERSION=1.0.0&IDENTIFIER=BufferPy
```

Listing 5.3: *DescribeProcess* request for *IntersectionPy*

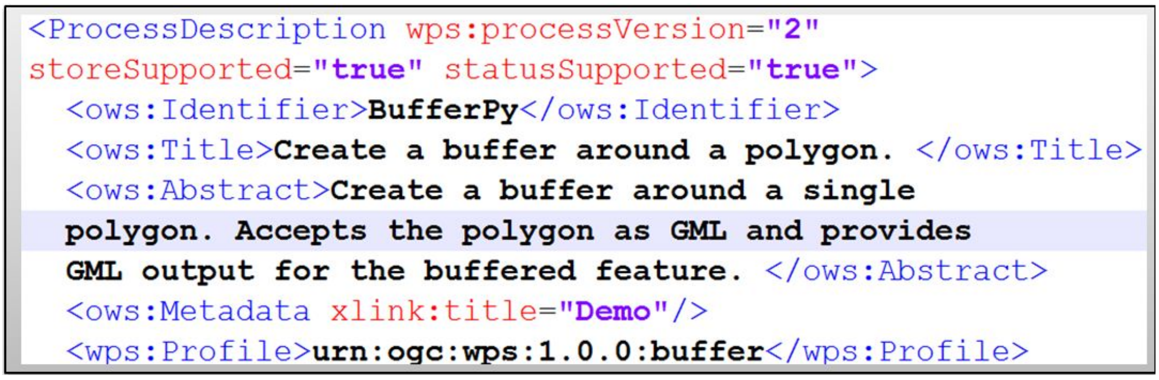
```
http://130.89.209.216/cgi-bin/zoo_loader.cgi?REQUEST=DescribeProcess&SERVICE=WPS&
VERSION=1.0.0&IDENTIFIER=IntersectionPy
```

5.4.1 Mapping *DescribeProcess* responses to chaining schema

This step is the process of selecting all required values from *DescribeProcess* responses content to the chaining schema. The selection of required values from *DescribeProcess* responses and identification of corresponding matching element in the chaining schema is determined by interpreting their elements.

The *DescribeProcess* responses of *BufferPy* and *IntersectionPy* must be specified as source schema, and chaining schema must be specified as target schema. For this we used one of schema mapping software, Atlova Map force, which has graphical interface to establish the mapping rules, the lines connecting elements, between the selected values of *DescribeProcess* response as shown on appendix, figure 1.

For example, figure 5.3, which is segment part of figure 1 which is shown in appendix, shows mapping of elements from *DescribeProcess* response of *BufferPy*, figure 5.2, to the corresponding elements of chaining schema, figure 5.3. On figure 5.4, the values of *storeSupported*, *storeSupported*, and identifier from figure 5.2 are copied to chaining schema, *storeSupported*, *storeSupported*, and identifier of the chaining schema, respectively. The remaining elements of the chaining schema, BaseURL, and *PostionInchain* are elements which are not available from *DescribeProcess* response, rather required to be filled by the user.



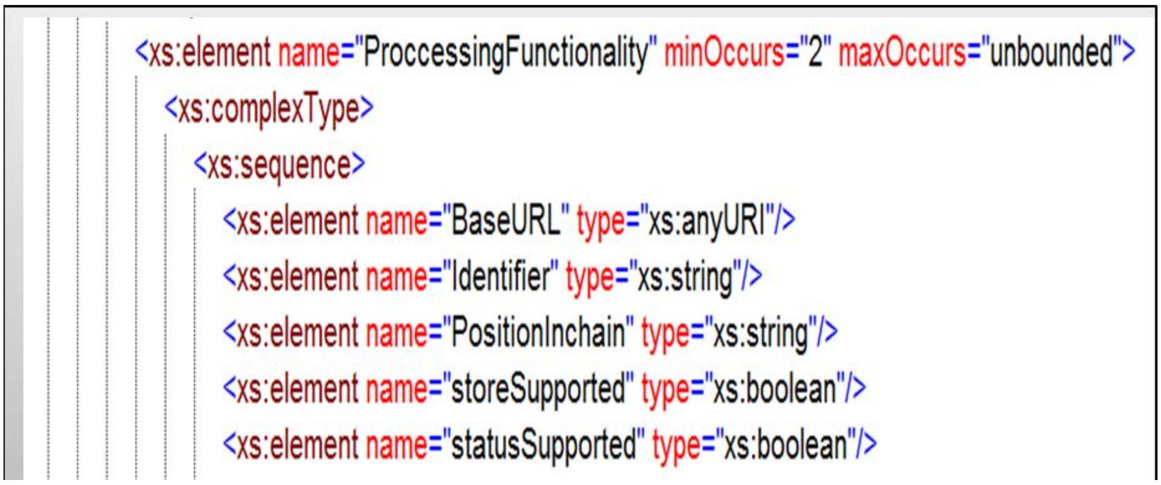
```
<ProcessDescription wps:processVersion="2"
storeSupported="true" statusSupported="true">
  <ows:Identifier>BufferPy</ows:Identifier>
  <ows:Title>Create a buffer around a polygon. </ows:Title>
  <ows:Abstract>Create a buffer around a single
  polygon. Accepts the polygon as GML and provides
  GML output for the buffered feature. </ows:Abstract>
  <ows:Metadata xlink:title="Demo"/>
  <wps:Profile>urn:ogc:wps:1.0.0:buffer</wps:Profile>
```

Figure 5.2: Segment of *DescribeProcess* response describing *BufferPy*

5.4.2 Mapping rules implementation

Once the correspondence elements between *DescribeProcess* response and chaining schema were determined, the corresponding mapping rules established, the XSLT code is generated which is one of the XML based language integrated with the software. The XSLT automatically interprets and assigns those selected values to the corresponding elements of chaining schema. The full generated XSLT code which is applicable specifically for this scenario is provided in appendix, listing 6. Figure 5.5 shows the corresponding XSLT code encoding the established mapping rules on figure 5.4 and figure 5.6 shows the final output after the required values are copied using the XSLT code.

The final result of this step is a single XML document that contains the two processing-functionalities required information which are ready to be executed which is shown in Listing 5.4.



```
<xs:element name="ProccessingFunctionality" minOccurs="2" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="BaseURL" type="xs:anyURI"/>
      <xs:element name="Identifier" type="xs:string"/>
      <xs:element name="PositionInchain" type="xs:string"/>
      <xs:element name="storeSupported" type="xs:boolean"/>
      <xs:element name="statusSupported" type="xs:boolean"/>
```

Figure 5.3: Segment elements of chaining schema

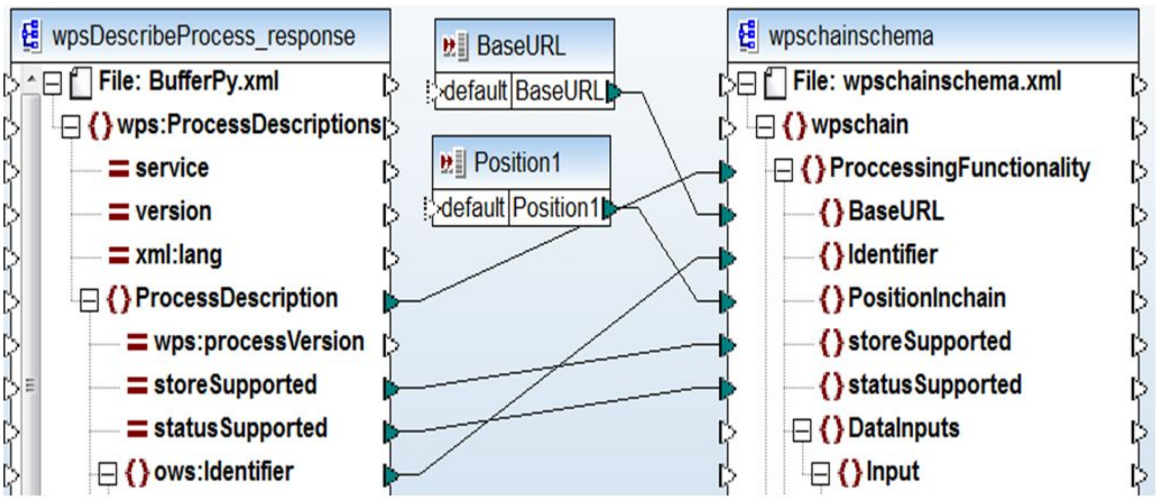


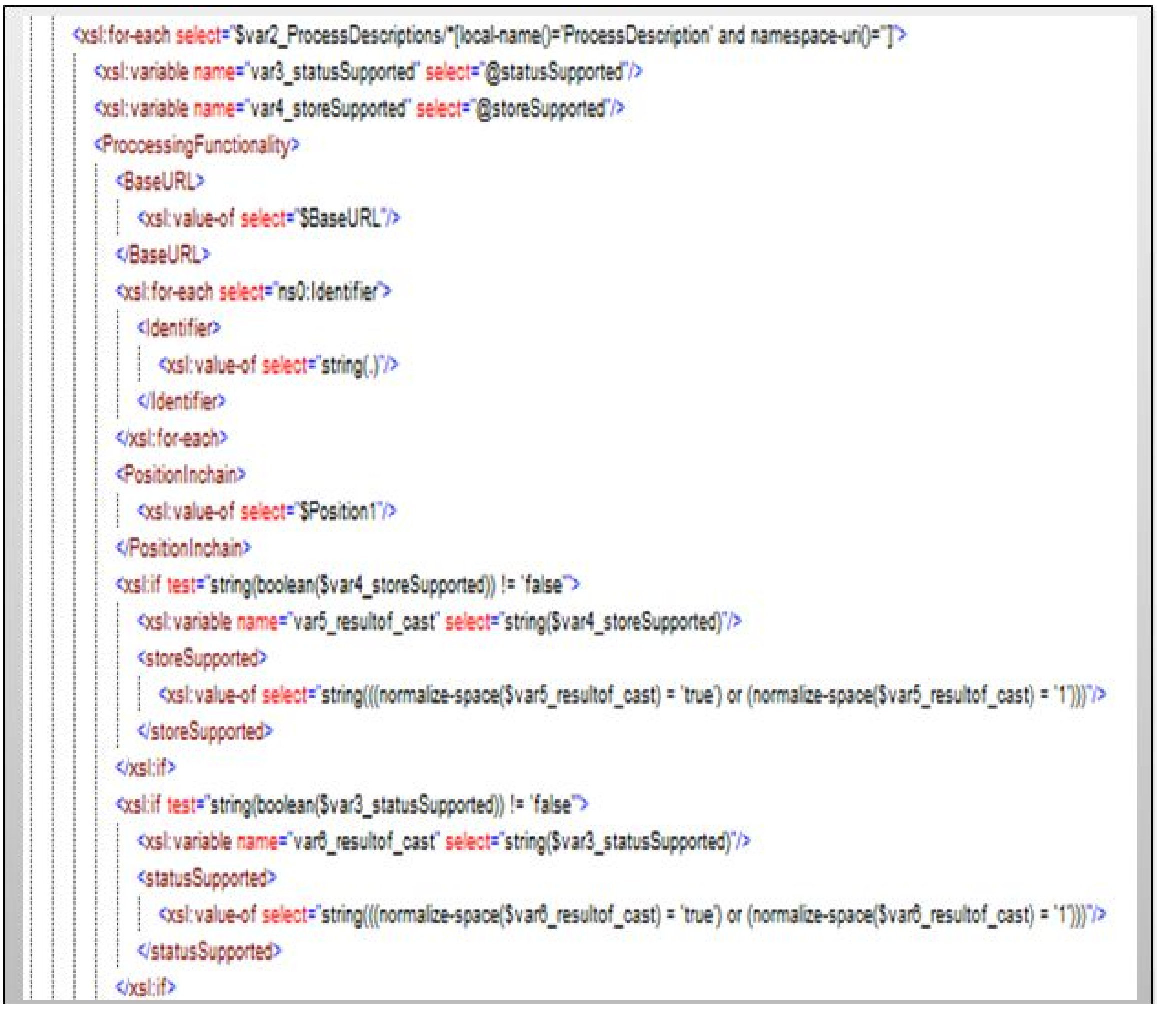
Figure 5.4: Segment of mapping *DescribeProcess* response of *BufferPy* to chaining schema

5.4.3 Chaining schema implementation

The designed chaining schema is given in section 4.2.3. The implementation of this chaining schema is demonstrated using the previously analyzed chaining requirement. Listing 5.4 shows that the two processing-functionalities are chained in a single XML document based on the chaining schema.

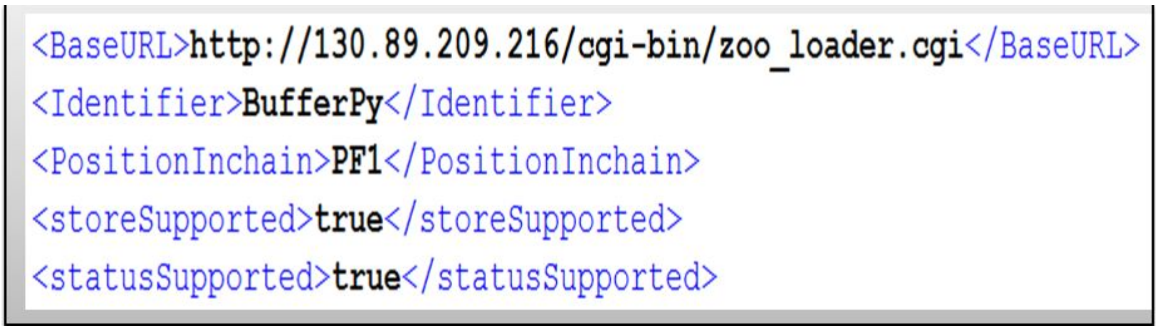
In this chaining example, the first processing-functionality is *BufferPy* where all required parameters and values for its execution within this chain instance are defined from line 6 to line 31. The *BufferPy* can be uniquely identified by the value of *PositionInchain*, *PF1*, on line 9. The *PF1* represents "first processing-functionality" to indicate as it is the first processing-functionality, in sequence of this chain.

In addition, as we can see, the *BufferPy* takes two input values which are defined from line 12 to line 25. The first input is defined from line 14 to line 20, with input identifer *Inputpolygon*. For *Inputpolygon*, the input value is reference (URL of WFS *GetFeature* request for road data) which



```
<xsl:for-each select="$var2_ProcessDescriptions/*[local-name()='ProcessDescription' and namespace-uri()='']">
  <xsl:variable name="var3_statusSupported" select="@statusSupported"/>
  <xsl:variable name="var4_storeSupported" select="@storeSupported"/>
  <ProcessingFunctionality>
    <BaseURL>
      <xsl:value-of select="$BaseURL"/>
    </BaseURL>
    <xsl:for-each select="ns0:Identifier">
      <Identifier>
        <xsl:value-of select="string(.)"/>
      </Identifier>
    </xsl:for-each>
    <PositionInchain>
      <xsl:value-of select="$Position1"/>
    </PositionInchain>
    <xsl:if test="string(boolean($var4_storeSupported)) != 'false'">
      <xsl:variable name="var5_resultof_cast" select="string($var4_storeSupported)"/>
      <storeSupported>
        <xsl:value-of select="string(((normalize-space($var5_resultof_cast) = 'true') or (normalize-space($var5_resultof_cast) = '1')))"/>
      </storeSupported>
    </xsl:if>
    <xsl:if test="string(boolean($var3_statusSupported)) != 'false'">
      <xsl:variable name="var6_resultof_cast" select="string($var3_statusSupported)"/>
      <statusSupported>
        <xsl:value-of select="string(((normalize-space($var6_resultof_cast) = 'true') or (normalize-space($var6_resultof_cast) = '1')))"/>
      </statusSupported>
    </xsl:if>
```

Figure 5.5: Segment of XSLT code used for implementing of resulted mapping rules of figure 5.4



```
<BaseURL>http://130.89.209.216/cgi-bin/zoo_loader.cgi</BaseURL>
<Identifier>BufferPy</Identifier>
<PositionInchain>PF1</PositionInchain>
<storeSupported>true</storeSupported>
<statusSupported>true</statusSupported>
```

Figure 5.6: Segment result of the final output after mapping rules executed

is one of the input type among the four input types defined in the generic chaining schema. The second input is defined from line 21 to line 24, with input identifer *BufferDistance* and input value is literal input which is the value for the road expansion. The *BufferPy* also produces one output with identifer *Result* which is defined from line 26 to 30. This output identifier and *PositionIn-*

chain, *PF1*, are defined as input for the second processing-functionality as we can see from line 41 to 44. So that output of the *BufferPy* can be identified with this values to be passed as input for the next processing-functionality.

With the same approach, the second processing-functionality, *IntersectionPy*, is defined from line 32 to 60. The *IntersectionPy* takes two input values which are defined from line 38 to 54. The first input with identifer, *InputEntity1*, which is defined from line 39 to 45, takes the output of *BufferPy*. For this, as we discussed in previous paragraph, the *PositionInchain* and output identifier of *BufferPy* are presented as input value. The second input value, with identifer *InputEntity2*, is reference input which is the URL of WFS *GetFeature* request for parcels data. The *IntersectionPy* also produces one output with identifer *Result* which is defined from line 55 to 59.

The complete output of the two processing-functionalities is the output of the last executed processing-functionalities which is the output of *IntersectionPy*. For this the *PositionInchain* and the output identifier of *IntersectionPy* are assigned as final output of the chain which defined on line 62 and 63.

Listing 5.4: A sample of chaining buffer and intersection processing-functionalities chained using the designed chaining schema

```
1      <?xml version="1.0" encoding="UTF-8"?>
2  <wpschain xmlns="http://www.itc.nl/wpschain" xmlns:xsi=
3  "http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://www.itc.nl/wpschain␣M:/
5  implemetation/FINALI~1/wpschainschema.xsd">
6          <ProccessingFunctionality>
7                  <BaseURL>http://130.89.209.216/cgi-bin/zoo_loader.cgi</BaseURL>
8                  <Identifier>BufferPy</Identifier>
9                  <PositionInchain>PF1</PositionInchain>
10                 <storeSupported>true</storeSupported>
11                 <statusSupported>true</statusSupported>
12                 <DataInputs>
13                         <Input>
14                                 <Identifier>InputPolygon</Identifier>
15                                 <Reference>http://130.89.209.216/cgi-bin/
16                                 mapserv?MAP=/home/students/meaza/
17                                 enschede/wps.map&amp;service=WFS&amp;
18                                 version=1.0.0&amp;request=
19                                 Getfeature&amp;typename=hengelosestraatWFS</Reference>
20                        </Input>
21                         <Input>
22                                 <Identifier>BufferDistance</Identifier>
23                                 <Litral>25</Litral>
24                         </Input>
25                 </DataInputs>
26                 <ProcessOutputs>
27                         <Output>
28                                 <Identifier>Result</Identifier>
29                         </Output>
30                 </ProcessOutputs>
31         </ProccessingFunctionality>
32         <ProccessingFunctionality>
33                 <BaseURL>http://130.89.209.216/cgi-bin/zoo_loader.cgi</BaseURL>
34                 <Identifier>IntersectionPy</Identifier>
35                 <PositionInchain>PF2</PositionInchain>
36                 <storeSupported>true</storeSupported>
37                 <statusSupported>true</statusSupported>
38                 <DataInputs>
39                         <Input>
```

```
                                <Identifier>InputEntity1</Identifier>
                                <ProcessedResult>
                                        <PositionInchain>PF1</PositionInchain>
                                        <OutPutIdentifier>Result</OutPutIdentifier>
                                </ProcessedResult>
                        </Input>
                        <Input>
                                <Identifier>InputEntity2</Identifier>
                                <Reference>http://130.89.209.216/cgi-bin/
                                mapserv?MAP=/home/students/meaza/enschede/
                                wps.map&amp;service=WFS&amp;
                                version=1.0.0&amp;request=Getfeature&amp;
                                typename=enschedeWFS</Reference>
                        </Input>
                </DataInputs>
                <ProcessOutputs>
                        <Output>
                                <Identifier>Result</Identifier>
                        </Output>
                </ProcessOutputs>
        </ProccessingFunctionality>
        <ChainOutput>
                <PositionInchain>PF2</PositionInchain>
                <OutputIdentifier>Result</OutputIdentifier>
                <ChainOutputIdentifier>BufferredIntersectedOutput</ChainOutputIdentifier
        </ChainOutput>
</wpschain>
```

5.4.4 Chaining engine implementation

In this step, final step, the result of the previous steps, the single XML document containing the two processing-functionalities based on their predefined order needs to be executed. For this purpose, the chaining engine was designed as given 4.3.7. The purpose of the chaining engine is to interpret and execute any chaining instance which contains any number of processing-functionalities designed based on the chaining schema. However, due to time limitation, for demonstration purpose the implemented engine is works only for this specific scenario. The implemented part of chaining engine is provided in appendix, on figure 2, figure 3, and figure 4. The returned out from executing chaining instance is in GML format and visualized as shown on figure 5.9. As we discussed, in section 5.2, the objective of the scenario was to search each parcels of Enschede shown on figure 5.7 which will be affected by the road expansion. In order to search these parcels, the *IntersectionPy* processing-functionality accepts a single polygon as input. Therefore, in order to execute the intersection of each parcels with the buffered road , the *IntersectionPy* needs to be executed iteratively. Due to this, we chose to demonstrate by taking Enschede as a single polygon as shown on figure 5.8 when the buffered road is intersected with this polygon , the result returns the buffered road itself as shown on figure 5.9.

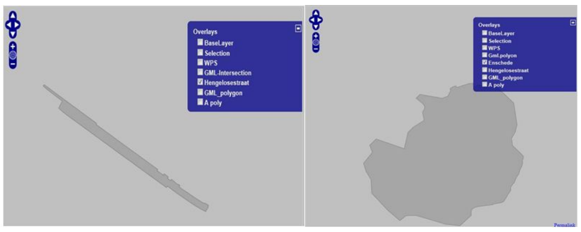


Figure 5.8: Road and Enschede polygon

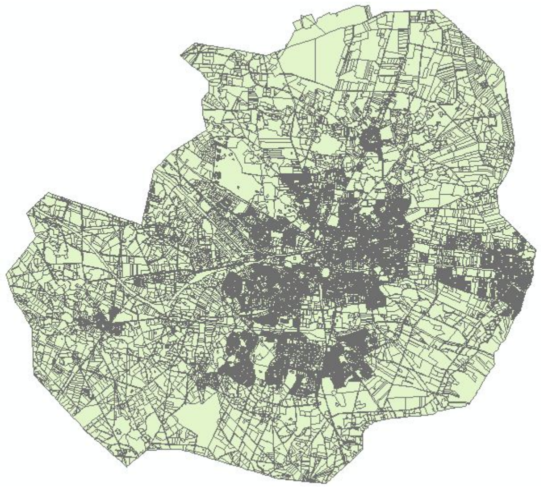


Figure 5.7: Enschede parcels

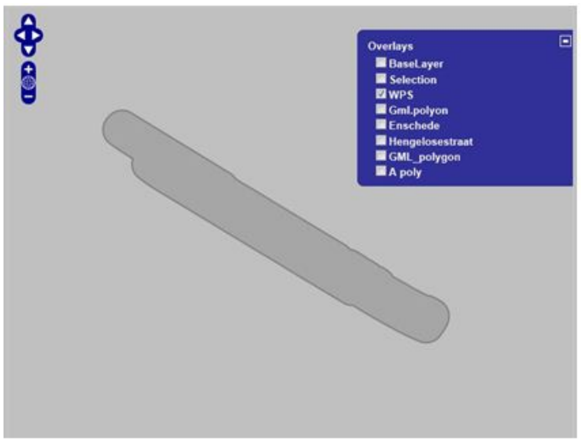


Figure 5.9: Buffered road intersected with Enschede polygon

5.5 SUMMARY

In this chapter we implemented and tested the concept of this research with an example as proof-of-concept. The chaining schema was applied for its intended purpose using the chosen scenario. The corresponding chaining engine implementation was also attempted and worked for demonstration purpose, although for its full functionality, all required cases needs to be implemented.

Chapter 6
Discussion, Conclusion, and Recommendation

In this chapter, we summarized the main objective of this research in terms of the research questions which were raised in chapter 1. We discussed the research questions and tried to answer each question in section 6.1. In addition, we offer a brief conclusion in section 6.2. Finally, we give few recommendations on what we think can be improved in future in section 6.3.

6.1 DISCUSSION ON RESEARCH QUESTIONS

The main objective of this research was to design a generic chaining method which enables to combine two or more number of disparate WSP processing-functionalities into service chain to perform complex geoprocessing task. To achieve this objective, five research questions were raised in section 1.2.2. In this section we discussed the achievements on those research questions.

1. What are the requirements to create service chain ?

   To create service chain, the description of each service is required. In service chain, the interaction of two or more number services is determined with their description. Usually, the content of services' description is comprised of functional and non-functional properties about the service. The functional properties describes about the functionality provided by the service which answers the question "what the service does ?" whereas the non-functional properties answers the question "how a services does ?". The functional properties include:service functionality, expected input parameters, and output parameters to be generated. The non-functional requirements include, QoS (quality of service) such as availability, response time, security, and cost. In service chain, the functional properties are the primarily needed requirements, because with out functional properties the service cannot exist, unlike the non-functional properties. In addition, in creating service chain, the chainability of participant services needs to evaluated. As we discussed in services chaining requirements analysis in chapter 3, the chainability evaluation can be determined by matching the output and input of each service. The evaluation can be done either using a metric unit that can measure the degree of similarity by parsing their input and output description with some specified threshold automatically, or a expert user can determine by reviewing their input and output description. Therefore based on the functional properties of services to create service chain, in a sequence of services, one service can be before another services to be executed, if and only if the output of the former service can satisfy the input requirements of the later service either completely or partially.

2. How to discover services to create service chain ?

   To start creating service chain, discovering and selecting chainable processing- functionalities is required. In order to discover available processing-functionalities, web address of WPS server, base URL, needs to be known. This base URL can be retrieved from catalogue services. Using this base URL, list of processing-functionalities within a single WPS server can be retrieved via *GetCapabilities* operation request. To select chainable processing-functionalities which fits a given task, the *DescribeProcess* responses should be retrieved using the base URL and name (identifer) discovered in the *GetCapabilities* response.

In case of more than one processing-functionalities are discovered with similar functionality properties, the ranking can be done either by human user giving his relevance weight or using software application. Here the non-functional properties can play key role to rank and select the best fittest processing-functionalities. In other case, if no processing-functionalities found that fulfill requirement of a given task, then the requirement of a given task needs to be refined into sub requirements until a set of processing-functionalities are discovered. However, due to missing semantics (meaning of inputs and outputs) description in *DescribeProcess* response document, one should be aware of that the wrong processing-functionality may be discovered which can be solved with semantic description as stated in [19].

3. How to assemble a set of services as service chain ?

   To assemble a set of processing-functionalities from different WPS provider into service chain, it is necessary to model the essential functional properties which enable two or more processing-functionalities interact with each other within a chain. We captured these functional properties from the *DescribeProcess* response content as well as the requirements to create web service and geoservices chaining application domain. We designed a generic chaining schema using XSD language in section 4.2.3 which contains all possible contents required to create service chain conceptually, serving as a template, to chain any two or more number of WPS processing-functionalities as one application in any GI application domain. The included functional properties are:

   - Two or more number of processing-functionalities can be involved in a chain
   - Each participant processing-functionality must have a base URL (Uniform Resource Locator), and name (identifer).
   - Each participant processing-functionality must have a position identifer within a chain.
   - Each processing-functionality should have information on the store supported and status supported as optional parameters.
   - Each processing-functionality in the chain can take more than one input and produce more than outputs.
   - Each input and output must have a unique identifer.
   - The output of one processing-functionality can be passed as input to the next processing-functionality.
   - For each processing-functionality input and output value can passed by value or reference (URL).
   - The final chain output is the output of the last executed processing-functionality.

4. How to present the service chain as a single service ?

   Any two or more number of WPS processing-functionalities chained using the designed chaining schema can be presented by single XML document, as a single application, which has a set of processing-functionalities with all required parameters and values such as base URL, name (identifer), input identifier and the corresponding input values, and output identifer of one processing-functionality to be passed as input for another processing-functionality based on specified order of execution. This XML document can be interpreted and executed by the chaining engine, which controls the execution request of each processing-functionality as if the user is dealing with a single processing-functionalities only.

5. How to coordinate or control the interaction among a set of participant services for input/output messages exchange (i.e. how the output of one service can be used as input for the other services) ?

   Any processing-functionalities chain instance created based on the generic chaining schema is interpreted and executed by the chaining engine to produce the result. The chaining engine makes the processing-functionalities working by sending an execute request to each processing-functionality one after another, by passing the output of one processing- functionality as input of the next processing-functionality to be executed. The returned output from one processing-functionality can be stored in a temporary variable holder which can be defined in the chaining engine, and this variable can passed as input to the next processing-functionality to be executed. If the processing-functionality supports output storage, then this processing-functionality can be requested to return the output by reference and this reference can be passed as input to the next processing-functionality.

6.2 CONCLUSION

In this research, we proposed a generic chaining schema to chain WPS processing-functionalities in a distributed web service-based environment. The chaining schema is generic by containing all necessary information which enable to create any two or more number of processing-functionalities in any GI application. To design such generic chaining schema, we analyzed the requirements to create service chain from web services and geoservices domains, and *DescribeProcess* response content. From service chaining requirement analysis, we knew that the description, which is comprised of functional and non-functional properties about a service is required to create service chain. We considered the functional properties as they are the fundamental requirements to use a service. From the *DescribeProcess* response content analysis, the mandatory and optional functional properties (parameters) were selected. Using the two analysis as input, we were able to design a generic chaining schema using XSD language as shown on figure 4.2. In addition, after we completed the design of this chaining schema, making ready for use, an architecture for chaining of WPS processing-functionalities using this chaining schema is provided, as shown on figure 4.6.

Any two or more number of WPS processing-functionalities chained using this chaining schema can be presented by single XML document, as one single application, which contains a number of processing-functionalities with all required information and order execution. In order to interpret and execute this XML document, we designed a chaining engine in section 4.3.7. The objective of this chaining engine to read the content of any number of processing-functionalities chained based on the chaining schema and to trigger the execution of each processing-functionality based on their defined order and provided information. As a proof-of-concept, a real-world scenario was chosen and demonstrated using the designed chaining schema. The output of this research can used as one of the standards in the OGC framework. The output aims at supporting expert users of GI application domain for utilization of two or more number of disparate WPS processing-functionalities as a single application.

6.3 RECOMMENDATION

The following recommendations were formulated to improve on the outcome of this research for future.

- Automation requirement: The chaining of processing-functionalities using our chaining schema is expert user driven. Because, the discovery of appropriate processing-functionalities for the required task, evaluation of the validity of inputs for each processing-functionality, the output and input compatibility for each processing-functionality, and evaluating the correctness of the chain result are remained issues on the expert user side. This can be improved through semantics description of processing-functionalities and through ontology classification of processing-functionalities. So that by reasoning processing-functionalities can be selected automatically to be chained.

- Auxiliary processing-functionalities requirement: The integration of auxiliary processing-functionalities should be included to the chaining engine in order to handle intermediate input and output format transformation such as GML to shape file, JPG , PNG, or SVG to visualize the result on the-fly, and coordinate transformation service to handle the conversion of from one spatial reference system to another incase of input and output are in different coordinate system. By auxiliary, to mean service integrated with the chaining engine for intermediate processing task requirements which are not identified in the beginning of the creating chaining instance to be executed.

- Chain instance reusability requirement: Once a chain of processing-functionalities has been developed for some user requirements, this chain can also exposed as new WPS processing-functionalities. So that for future need, already built chain can be used for more than one similar application requirement rather than building the new chain starting from the scratch. By this, the central concept of SOA, service reusability can be utilized. This can be achieved by publishing the processing-functionalities chain via WPS interface, so that it can accessed via *GetCapabilities*, *DescribeProcess*, and *Execute* operations in a similar way of other atomic WPS processing-functionalities.

LIST OF REFERENCES

[1] Web Services Business Process Execution Language(BPEL), Version 2.0. `http://docs.oasis-open.org/wsbpel/2.0/CS01/wsbpel-v2.0-CS01.html`, 2007. accessed 21-May-2011.

[2] Geographic information - services, iso tc211 document number n1203. `http://www.isotc211.org`, 2008. accessed 10-November-2011.

[3] ISO 19119 and OGC Service Architecture. The opengis abstract specification topic 12: Opengis service architecture, version 4.3, 2002, 2004.

[4] N. Alameh. Chaining geographic information web services. *IEEE Internet Computing*, pages 22–29, 2003.

[5] N. Alameh. Service chaining of interoperable geographic information web services. *IEEE Internet Computing*, pages 22–29, 2003.

[6] Z. Azmeh, M. Driss, F. Hamoui, M. Huchard, N. Moha, and C. Tibermacine. Selection of composable web services driven by user requirements. In *Web Services (ICWS),IEEE International Conference on*, pages 395–402. IEEE, 2011.

[7] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, M. Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, et al. Universal Description Discovery Integration(UDDI), Version 3.0. *Oasis*, 2002.

[8] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple object access protocol (SOAP) 1.1. `http://www.w3.org/TR/2000/NOTE-SOAP-20000508/`, 2000. accessed 21-May-2011.

[9] J. Brauner, T. Foerster, B. Schaeffer, and B. Baranski. Towards a research agenda for geoprocessing services. In *12th AGILE International Conference on Geographic Information Science*, 2009.

[10] E. Christensen, F. Curbera, G. Meredith, and Weerawarana.S. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/2001/NOTE-wsdl-20010315`, 2001. accessed 21-May-2011.

[11] T. Foerster, B. Schäffer, B. Baranski, and J. Brauner. Geospatial web services for distributed processing-applications and scenarios. 2011.

[12] A. Friis-Christensen, N. Ostlander, M. Lutz, and L. Bernard. Designing service architectures for distributed geoprocessing: Challenges and future directions. *Transactions in GIS*, pages 799–818, 2007.

[13] N. Josuttis. *SOA in practice:The art of distributed system design*. 2007.

[14] C. Kiehle. Business logic for geoprocessing of distributed geodata. *Computers Geosciences*, pages 1746–1757, 2006.

[15] C. Kiehle, K. Greve, and C. Heier. Requirements for next generation spatial data infrastructures-standardized web based geoprocessing and web service orchestration. *Transactions in GIS*, pages 819–834, 2007.

[16] L. Lehto and T. Sarjakoski. Schema translations by xslt for gml-encoded geospatial data in heterogeneous web-service environment. In *Proceedings of the XXth ISPRS Congress, July*, pages 12–23, 2004.

[17] R. Lemmens, D. Rolf, W. Andreas, G. Carlos, G. Michael, and V.O. Peter. Integrating semantic and syntactic descriptions to chain geographic services. *IEEE Internet Computing*, pages 42–52, 2006.

[18] R. Lemmens, D. Rolf, W. Andreas, G. Carlos, G. Michael, and V.O. Peter. Enhancing geo-service chaining through deep service descriptions. *Transactions in GIS*, pages 849–871, 2007.

[19] M. Lutz. Ontology-based descriptions for semantic discovery and composition of geoprocessing services. *Geoinformatica*, pages 1–36, 2007.

[20] C. Michaelis and D. Ames. Evaluation and implementation of the ogc web processing service for use in client-side gis. *Geoinformatica*, pages 109–120, 2009.

[21] J. Mukherjee and S. Ghosh. Geospatial service chaining in decision support systems. In *India Annual Conference*, pages 1–4, 2010.

[22] SOA OASIS. Reference model tc, reference model for service oriented architecture 1.0. Technical report, 2006.

[23] J. Rao and X. Su. A survey of automated web service composition methods. *Semantic Web Services and Web Process Composition*, 2005.

[24] B. Schäffer. Ws-6 geoprocessing workflow (gpw) version 0.3.0, architecture engineering report,ogc 09-053r5. `http://www.opengeospatial.org/standards/per`, 2009. accessed 21-May-2011.

[25] P. Schut. OGC Web Processing Service (WPS), verion 1.0.0, OGC Standard Document. `http://www.opengeospatial.org/standards/wps/`, 2007.

[26] B. Stollberg and A. Zipf. Ogc web processing service interface for web service orchestration aggregating geo-processing services in a bomb threat scenario. *Web and Wireless Geographical Information Systems*, pages 239–251, 2007.

[27] D. Than. Web service orchestration. `http://www.eurescom.de/message/messageJun2003/Web_Service_Orchestration.asp`, 2010. accessed 21-May-2011.

[28] S.J. Vaughan-Nichols. Web services: Beyond the hype. *Computer*, pages 18–21, 2002.

[29] A. Weiser and A. Zipf. Web service orchestration of ogc web services for disaster management. *Geomatics Solutions for Disaster Management*, pages 239–254, 2007.

[30] C. Yang and R. Raskin. Introduction to distributed geographic information processing research. *International Journal of Geographical Information Science*, pages 553–560.

[31] D. Zhang, B. Xie, and L. Di. Open geospatial information services chaining based on ogc specifications and processing model. In *Education Technology and Training.International Workshop on Geoscience and Remote Sensing.International Workshop on*, pages 153–157, 2008.

Appendix

Listing 1: Example *GetCapabilities* response from Zoo WPS

```
<?xml version="1.0" encoding="utf-8"?>
<wps:Capabilities xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
    http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd"
    service="WPS" xml:lang="en-US" version="1.0.0">
  <ows:ServiceIdentification>
    <ows:Title>Zoo WPS Test Server</ows:Title>
    <ows:Abstract>Development version of ZooWPS. See
    http://www.zoo-project.org</ows:Abstract>
    <ows:Keywords>
      <ows:Keyword>WPS</ows:Keyword>
      <ows:Keyword>GIS</ows:Keyword>
      <ows:Keyword>buffer</ows:Keyword>
    </ows:Keywords>
    <ows:ServiceType>WPS</ows:ServiceType>
    <ows:ServiceTypeVersion>1.0.0</ows:ServiceTypeVersion>
    <ows:Fees>None</ows:Fees>
    <ows:AccessConstraints>none</ows:AccessConstraints>
  </ows:ServiceIdentification>
  <ows:ServiceProvider>
    <ows:ProviderName>GIP Department</ows:ProviderName>
    <ows:ProviderSite xlink:href="http://130.89.209.216"/>
    <ows:ServiceContact>
      <ows:IndividualName>Javier MORALES</ows:IndividualName>
      <ows:PositionName>Teacher</ows:PositionName>
      <ows:ContactInfo>
        <ows:Phone>
          <ows:Voice>False</ows:Voice>
          <ows:Facsimile>False</ows:Facsimile>
        </ows:Phone>
        <ows:Address>
          <ows:DeliveryPoint>Hengelosestraat 99</ows:DeliveryPoint>
          <ows:City>Enschede</ows:City>
          <ows:AdministrativeArea>Netherlands</ows:AdministrativeArea>
          <ows:PostalCode>7514AE</ows:PostalCode>
          <ows:Country>nl</ows:Country>
          <ows:ElectronicMailAddress>email@itc.nl</ows:ElectronicMailAddress>
        </ows:Address>
      </ows:ContactInfo>
    </ows:ServiceContact>
  </ows:ServiceProvider>
  <ows:OperationsMetadata>
    <ows:Operation name="GetCapabilities">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://130.89.209.216//"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="DescribeProcess">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://130.89.209.216//"/>
          <ows:Post xlink:href="http://130.89.209.216//"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
    <ows:Operation name="Execute">
      <ows:DCP>
        <ows:HTTP>
          <ows:Get xlink:href="http://130.89.209.216//"/>
          <ows:Post xlink:href="http://130.89.209.216//"/>
        </ows:HTTP>
      </ows:DCP>
    </ows:Operation>
  </ows:OperationsMetadata>
  <wps:ProcessOfferings>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>CentroidPy</ows:Identifier>
      <ows:Title>Get the centroid of a polygon. </ows:Title>
      <ows:Abstract>Compute the geometry centroid.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="1">
      <ows:Identifier>Boundary</ows:Identifier>
      <ows:Title>Compute boundary.</ows:Title>
      <ows:Abstract>A new geometry object is created and returned containing
    the boundary of the geometry on which the method is invoked. </ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>BufferPy</ows:Identifier>
      <ows:Title>Create a buffer around a polygon. </ows:Title>
      <ows:Abstract>Create a buffer around a single polygon. Accepts the
    polygon as GML and provides GML output for the buffered feature.
    </ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>SymDifferencePy</ows:Identifier>
      <ows:Title>Compute symmetric difference. </ows:Title>
      <ows:Abstract>Generates a new geometry which is the symmetric difference
    of this geometry and the other geometry.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="1">
      <ows:Identifier>BoundaryPy</ows:Identifier>
      <ows:Title>Compute boundary.</ows:Title>
      <ows:Abstract>A new geometry object is created and returned containing
    the boundary of the geometry on which the method is invoked. </ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>IntersectionPy</ows:Identifier>
      <ows:Title>Compute intersection. </ows:Title>
      <ows:Abstract>Generates a new geometry which is the region of
    intersection of the two geometries operated on.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>UnionPy</ows:Identifier>
      <ows:Title>Compute union. </ows:Title>
      <ows:Abstract>Generates a new geometry which is the region of union of
    the two geometries operated on.</ows:Abstract>
```

```
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="1">
      <ows:Identifier>ConvexHullPy</ows:Identifier>
      <ows:Title>Compute convex hull.</ows:Title>
      <ows:Abstract>A new geometry object is created and returned containing
    the convex hull of the geometry on which the method is
    invoked.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="1">
      <ows:Identifier>ConvexHull</ows:Identifier>
      <ows:Title>Compute convex hull.</ows:Title>
      <ows:Abstract>A new geometry object is created and returned containing
    the convex hull of the geometry on which the method is
    invoked.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>DifferencePy</ows:Identifier>
      <ows:Title>Compute difference. . </ows:Title>
      <ows:Abstract>Generates a new geometry which is the region of this
    geometry with the region of the other geometry removed.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>GetArea</ows:Identifier>
      <ows:Title>Compute geometry area.</ows:Title>
      <ows:Abstract>Computes the area for a geometry</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>Centroid</ows:Identifier>
      <ows:Title>Get the centroid of a polygon. </ows:Title>
      <ows:Abstract>Compute the geometry centroid.</ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
    <wps:Process wps:processVersion="2">
      <ows:Identifier>Buffer</ows:Identifier>
      <ows:Title>Create a buffer around a polygon. </ows:Title>
      <ows:Abstract>Create a buffer around a single polygon. Accepts the
    polygon as GML and provides GML output for the buffered feature.
    </ows:Abstract>
      <ows:Metadata xlink:title="Demo"/>
    </wps:Process>
  </wps:ProcessOfferings>
  <wps:Languages>
    <wps:Supported>
      <ows:Language>en-US</ows:Language>
    </wps:Supported>
  </wps:Languages>
</wps:Capabilities>
```

Listing 2: Example *DescribeProcess* response describing BufferPy from Zoo

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ProcessDescriptions xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
  http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
  service="WPS" version="1.0.0" xml:lang="en-US">
<ProcessDescription wps:processVersion="2" storeSupported="true"
  statusSupported="true">
  <ows:Identifier>BufferPy</ows:Identifier>
  <ows:Title>Create a buffer around a polygon. </ows:Title>
  <ows:Abstract>Create a buffer around a single polygon. Accepts the polygon
  as GML and provides GML output for the buffered feature. </ows:Abstract>
  <ows:Metadata xlink:title="Demo"/>
  <wps:Profile>urn:ogc:wps:1.0.0:buffer</wps:Profile>
  <DataInputs>
    <Input minOccurs="1" maxOccurs="1">
      <ows:Identifier>InputPolygon</ows:Identifier>
      <ows:Title>Polygon to be buffered</ows:Title>
      <ows:Abstract>URI to a set of GML that describes the
  polygon.</ows:Abstract>
      <ComplexData>
        <Default>
          <Format>
            <MimeType>text/xml</MimeType>
            <Encoding>UTF-8</Encoding>
            <Schema>http://schemas.opengis.net/gml/3.1.0/base/feature.xsd</Schema>
          </Format>
        </Default>
        <Supported>
          <Format>
            <MimeType>application/json</MimeType>
            <Encoding>UTF-8</Encoding>
          </Format>
        </Supported>
      </ComplexData>
    </Input>
    <Input minOccurs="0" maxOccurs="1">
      <ows:Identifier>BufferDistance</ows:Identifier>
      <ows:Title>Buffer Distance</ows:Title>
      <ows:Abstract>Distance to be used to calculate buffer.</ows:Abstract>
      <LiteralData>
        <ows:DataType
  ows:reference="http://www.w3.org/TR/xmlschema-2/#float">float</ows:DataType>
        <UOMs>
          <Default>
            <ows:UOM>meters</ows:UOM>
          </Default>
          <Supported>
            <ows:UOM>feet</ows:UOM>
          </Supported>
        </UOMs>
        <ows:AnyValue/>
        <DefaultValue>1</DefaultValue>
      </LiteralData>
    </Input>
  </DataInputs>
  <ProcessOutputs>
    <Output>
      <ows:Identifier>Result</ows:Identifier>
      <ows:Title>Buffered Polygon</ows:Title>
      <ows:Abstract>GML stream describing the buffered polygon
  feature.</ows:Abstract>
      <ComplexOutput>
```

```
            <Default>
              <Format>
                <MimeType>text/xml</MimeType>
                <Encoding>UTF-8</Encoding>
                <Schema>http://schemas.opengis.net/gml/3.1.0/base/feature.xsd</Schema>
              </Format>
            </Default>
            <Supported>
              <Format>
                <MimeType>application/json</MimeType>
                <Encoding>UTF-8</Encoding>
              </Format>
            </Supported>
          </ComplexOutput>
        </Output>
      </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing 3: Example of *DescribeProcess* response describing IntersectionPy from Zoo

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ProcessDescriptions xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
    http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
    service="WPS" version="1.0.0" xml:lang="en-US">
  <ProcessDescription wps:processVersion="2" storeSupported="true"
    statusSupported="true">
    <ows:Identifier>IntersectionPy</ows:Identifier>
    <ows:Title>Compute intersection. </ows:Title>
    <ows:Abstract>Generates a new geometry which is the region of intersection
    of the two geometries operated on.</ows:Abstract>
    <ows:Metadata xlink:title="Demo"/>
    <wps:Profile>urn:ogc:wps:1.0.0:union</wps:Profile>
    <DataInputs>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>InputEntity1</ows:Identifier>
        <ows:Title>Mon test  </ows:Title>
        <ows:Abstract>the first geometry to compare against.</ows:Abstract>
        <ComplexData>
          <Default>
            <Format>
              <MimeType>text/xml</MimeType>
              <Encoding>UTF-8</Encoding>
              <Schema>http://fooa/gml/3.1.0/polygon.xsd</Schema>
            </Format>
          </Default>
          <Supported>
            <Format>
              <MimeType>text/xml</MimeType>
              <Encoding>base64</Encoding>
              <Schema>http://fooa/gml/3.1.0/polygon.xsd</Schema>
            </Format>
          </Supported>
        </ComplexData>
      </Input>
      <Input minOccurs="1" maxOccurs="1">
```

```
          <ows:Identifier>InputEntity2</ows:Identifier>
          <ows:Title>Mon test  </ows:Title>
          <ows:Abstract>the other geometry to compare against.</ows:Abstract>
          <ComplexData>
            <Default>
              <Format>
                <MimeType>text/xml</MimeType>
                <Encoding>UTF-8</Encoding>
                <Schema>http://fooa/gml/3.1.0/polygon.xsd</Schema>
              </Format>
            </Default>
            <Supported>
              <Format>
                <MimeType>text/xml</MimeType>
                <Encoding>base64</Encoding>
                <Schema>http://fooa/gml/3.1.0/polygon.xsd</Schema>
              </Format>
            </Supported>
          </ComplexData>
        </Input>
      </DataInputs>
      <ProcessOutputs>
        <Output>
          <ows:Identifier>Result</ows:Identifier>
          <ows:Title>Mon test  </ows:Title>
          <ows:Abstract>A new geometry representing the intersection or NULL if
      there is no intersection or an error occurs.</ows:Abstract>
          <ComplexOutput>
            <Default>
              <Format>
                <MimeType>application/json</MimeType>
                <Encoding>UTF-8</Encoding>
              </Format>
            </Default>
            <Supported>
              <Format>
                <MimeType>text/xml</MimeType>
                <Encoding>UTF-8</Encoding>
                <Schema>http://fooa/gml/3.1.0/polygon.xsd</Schema>
              </Format>
            </Supported>
          </ComplexOutput>
        </Output>
      </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing 4: Example DescribeProcess response describing Intersection from Deegree

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
      xmlns:ows="http://www.opengis.net/ows/1.1"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="WPS"
      version="1.0.0" xml:lang="en"
      xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
      http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd">
    <ProcessDescription wps:processVersion="1.0.0" storeSupported="true"
      statusSupported="false">
```

```
<ows:Identifier>Intersection</ows:Identifier>
<ows:Title>Determining the intersection points between two GML
Geometries.</ows:Title>
<ows:Abstract>The intersection of two Geometries A and B is the set of all
points which lie in both A and B.</ows:Abstract>
<DataInputs>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>GMLInput1</ows:Identifier>
    <ows:Title>GMLInput1</ows:Title>
    <ComplexData>
      <Default>
        <Format>
          <MimeType>text/xml</MimeType>
          <Encoding>UTF-8</Encoding>
          <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
        </Format>
      </Default>
      <Supported>
        <Format>
          <MimeType>text/xml</MimeType>
          <Encoding>UTF-8</Encoding>
          <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
        </Format>
      </Supported>
    </ComplexData>
  </Input>
  <Input minOccurs="1" maxOccurs="1">
    <ows:Identifier>GMLInput2</ows:Identifier>
    <ows:Title>GMLInput2</ows:Title>
    <ComplexData>
      <Default>
        <Format>
          <MimeType>text/xml</MimeType>
          <Encoding>UTF-8</Encoding>
          <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
        </Format>
      </Default>
      <Supported>
        <Format>
          <MimeType>text/xml</MimeType>
          <Encoding>UTF-8</Encoding>
          <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
        </Format>
      </Supported>
    </ComplexData>
  </Input>
</DataInputs>
<ProcessOutputs>
  <Output>
    <ows:Identifier>Intersection</ows:Identifier>
    <ows:Title>Intersection</ows:Title>
    <ComplexOutput>
      <Default>
        <Format>
          <MimeType>text/xml</MimeType>
          <Encoding>UTF-8</Encoding>
          <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
        </Format>
      </Default>
      <Supported>
```

```
                <Format>
                    <MimeType>text/xml</MimeType>
                    <Encoding>UTF-8</Encoding>
                    <Schema>http://schemas.opengis.net/gml/3.1.1/base/gml.xsd</Schema>
                </Format>
            </Supported>
        </ComplexOutput>
      </Output>
    </ProcessOutputs>
  </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing 5: Example DescribeProcess response describing Buffer from Deegree

```
<?xml version="1.0" encoding="UTF-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
    xmlns:ows="http://www.opengis.net/ows/1.1"
    xmlns:ogc="http://www.opengis.net/ogc"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="WPS"
    version="1.0.0" xml:lang="en"
    xsi:schemaLocation="http://www.opengis.net/wps/1.0.0␣
    http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd">
  <ProcessDescription wps:processVersion="1.0.0" storeSupported="true"
    statusSupported="false">
    <ows:Identifier>Buffer</ows:Identifier>
    <ows:Title>Process for creating a buffer around a GML geometry.</ows:Title>
    <ows:Abstract>The purpose of this process is to create a buffer around an
    existing geometry with a buffer distance specified by the
    user.</ows:Abstract>
    <DataInputs>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>GMLInput</ows:Identifier>
        <ows:Title>GMLInput</ows:Title>
        <ComplexData>
          <Default>
            <Format>
              <MimeType>text/xml</MimeType>
              <Encoding>UTF-8</Encoding>
              <Schema>http://schemas.opengis.net/gml/3.1.1/base/geometryComplexes.xsd</Schema>
            </Format>
          </Default>
          <Supported>
            <Format>
              <MimeType>text/xml</MimeType>
              <Encoding>UTF-8</Encoding>
              <Schema>http://schemas.opengis.net/gml/3.1.1/base/geometryComplexes.xsd</Schema>
            </Format>
          </Supported>
        </ComplexData>
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>BufferDistance</ows:Identifier>
        <ows:Title>Buffer distance</ows:Title>
        <LiteralData>
          <ows:DataType
    ows:reference="http://www.w3.org/TR/xmlschema-2/#double">double</ows:DataType>
          <UOMs>
            <Default>
              <ows:UOM>unity</ows:UOM>
```

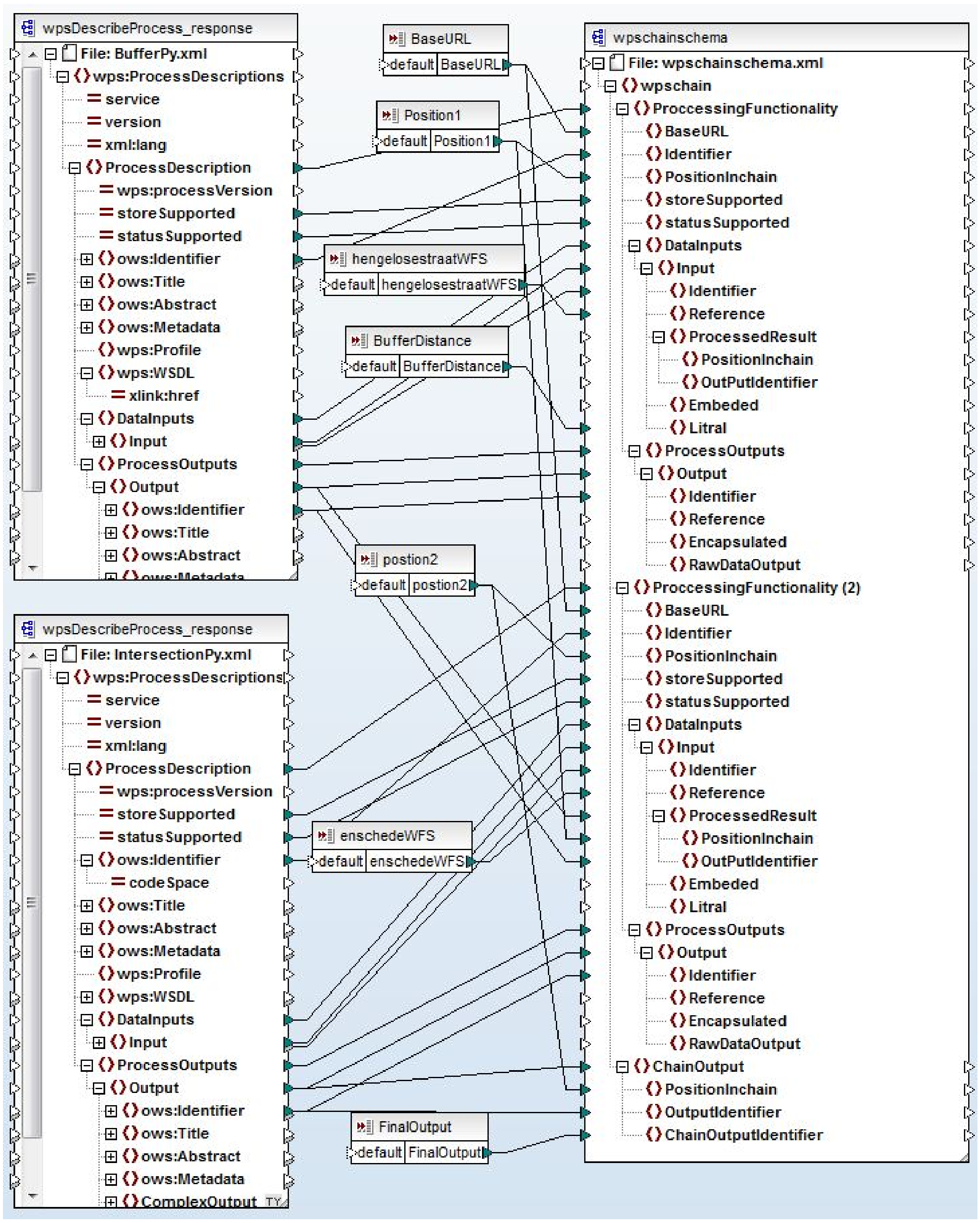


Figure 1: Schema mapping to fetch required value from DescribeProcess response of BufferPy and IntersectionPy to the chaining schema

```
                  </Default>
                  <Supported>
                     <ows:UOM>unity</ows:UOM>
                  </Supported>
               </UOMs>
               <ows:AnyValue/>
            </LiteralData>
         </Input>
      </DataInputs>
      <ProcessOutputs>
         <Output>
            <ows:Identifier>BufferedGeometry</ows:Identifier>
            <ows:Title>BufferedGeometry</ows:Title>
            <ComplexOutput>
               <Default>
                  <Format>
                     <MimeType>text/xml</MimeType>
                     <Encoding>UTF-8</Encoding>
                     <Schema>http://schemas.opengis.net/gml/3.1.1/base/geometryComplexes.xsd</Schema>
                  </Format>
               </Default>
               <Supported>
                  <Format>
                     <MimeType>text/xml</MimeType>
                     <Encoding>UTF-8</Encoding>
                     <Schema>http://schemas.opengis.net/gml/3.1.1/base/geometryComplexes.xsd</Schema>
                  </Format>
               </Supported>
            </ComplexOutput>
         </Output>
      </ProcessOutputs>
   </ProcessDescription>
</wps:ProcessDescriptions>
```

Listing 6: XSLT code for mapping rules implementation of *IntersectionPy* and *BufferPy* to the chaining scehma

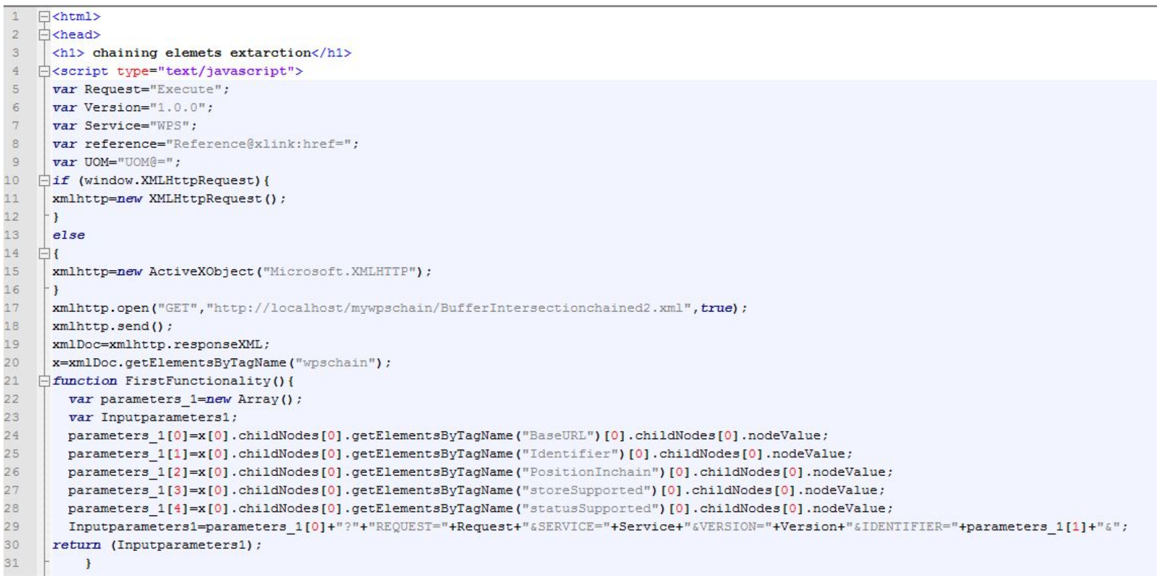
```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:ns0="http://www.opengis.net/ows/1.1" xmlns:ns1=
 "http://www.opengis.net/wps/1.0.0" xmlns:xs="http://www.w3.org/2001/
XMLSchema" xmlns:fn="http://www.w3.org/2005/xpath-functions"
 exclude-result-prefixes="ns0 ns1 xs fn">
        <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
        <xsl:param name="wpsDescribeProcess_response3" select="'IntersectionPy.xml'"/>
        <xsl:param name="BaseURL" as="xs:anyURI" required="yes"/>
        <xsl:param name="Position1" as="xs:string" required="yes"/>
        <xsl:param name="hengelosestraatWFS" as="xs:anyURI" required="yes"/>
        <xsl:param name="BufferDistance" as="xs:double" required="yes"/>
        <xsl:param name="postion2" as="xs:string" required="yes"/>
        <xsl:param name="enschedeWFS" as="xs:anyURI" required="yes"/>
        <xsl:param name="FinalOutput" as="xs:string" required="yes"/>
        <xsl:template match="/">
                <xsl:variable name="var1_ProcessDescriptions" as="node()?" select="fn:doc($wpsDescribeProcess_response3)
                /ns1:ProcessDescriptions"/>
                <xsl:variable name="var2_ProcessDescriptions" as="node()?" select="ns1:ProcessDescriptions"/>
                <wpschain xmlns="http://www.itc.nl/wpschain">
                        <xsl:attribute name="xsi:schemaLocation" namespace="http://www.w3.org/2001/XMLSchema-instance"
            select="'http://www.itc.nl/wpschain M:/implemetation/FINALI~1/wpschainschema.xsd'"/>
                        <xsl:for-each select="$var2_ProcessDescriptions/*:ProcessDescription[fn:namespace-uri() eq '']">
                                <xsl:variable name="var3_statusSupported" as="node()?"
                                        select="@statusSupported"/>
                                <xsl:variable name="var4_storeSupported" as="node()?"
                                        select="@storeSupported"/>
                                <ProccessingFunctionality>
                                        <BaseURL>
                                                <xsl:sequence
                                           select="xs:string($BaseURL)"/>
                                        </BaseURL>
                                        <xsl:for-each select="ns0:Identifier">
                                                <Identifier>
                                                        <xsl:sequence
                                               select="fn:string(.)"/>
                                                </Identifier>
                                        </xsl:for-each>
```

```
<PositionInchain>
    <xsl:sequence
  select="$Position1"/>
</PositionInchain>
<xsl:if test="fn:exists($var4_storeSupported)">
    <storeSupported>
        <xsl:sequence
  select="xs:string(xs:boolean(fn:string($var4_storeSupported)))"/>
    </storeSupported>
    </xsl:if>
<xsl:if test="fn:exists($var3_statusSupported)">
    <statusSupported>
        <xsl:sequence
  select="xs:string(xs:boolean(fn:string($var3_statusSupported)))"/>
    </statusSupported>
</xsl:if>
<xsl:for-each select="*:DataInputs[fn:namespace-uri() eq '']">
    <DataInputs>
        <xsl:for-each select="*:Input[fn:namespace-uri() eq '']">
            <Input>
                <xsl:for-each select="ns0:Identifier">
                    <Identifier>
                  <xsl:sequence
          select="fn:string(.)"/>
                    </Identifier>
                </xsl:for-each>
                <Reference>
                    <xsl:sequence
        select="xs:string($hengelosestraatWFS)"/>
                </Reference>
                <Litral>
                    <xsl:sequence
        select="xs:string($BufferDistance)"/>
                </Litral>
            </Input>
        </xsl:for-each>
    </DataInputs>
</xsl:for-each>
<xsl:for-each select="*:ProcessOutputs[fn:namespace-uri() eq '']">
```

```
            <ProcessOutputs>
                <xsl:for-each
                select="*:Output[fn:namespace-uri() eq '']">
                    <Output>
                        <xsl:for-each
                select="ns0:Identifier">
                            <Identifier>
                                <xsl:sequence
                   select="fn:string(.)"/>
                            </Identifier>
                        </xsl:for-each>
                    </Output>
                </xsl:for-each>
            </ProcessOutputs>
        </xsl:for-each>
    </ProccessingFunctionality>
</xsl:for-each>
<xsl:for-each select="$var1_ProcessDescriptions/*:ProcessDescription[fn:namespace-uri() eq '']">
    <xsl:variable name="var5_statusSupported" as="node()?"
     select="@statusSupported"/>
    <xsl:variable name="var6_storeSupported" as="node()?"
     select="@storeSupported"/>
    <ProccessingFunctionality>
        <BaseURL>
            <xsl:sequence
    select="xs:string($BaseURL)"/>
        </BaseURL>
        <xsl:for-each
      select="ns0:Identifier">
            <Identifier>
                <xsl:sequence
      select="fn:string(.)"/>
            </Identifier>
        </xsl:for-each>
        <PositionInchain>
            <xsl:sequence
        select="$postion2"/>
        </PositionInchain>
        <xsl:if test="fn:exists($var6_storeSupported)">
        <storeSupported>
            <xsl:sequence
select="xs:string(xs:boolean
            (fn:string($var6_storeSupported)))"/>
        </storeSupported>
    </xsl:if>
    <xsl:if test="fn:exists($var5_statusSupported)">
        <statusSupported>
            <xsl:sequence
select="xs:string(xs:boolean(fn:string
            ($var5_statusSupported)))"/>
        </statusSupported>
    </xsl:if>
    <xsl:for-each select="*:DataInputs[fn:namespace-uri() eq '']">
        <DataInputs>
            <xsl:for-each select="*:Input[fn:namespace-uri() eq '']">
                <Input>
                    <xsl:for-each select="ns0:Identifier">
                        <Identifier>
                            <xsl:sequence
           select="fn:string(.)"/>
                        </Identifier>
                    </xsl:for-each>
                    <Reference>
                        <xsl:sequence
            select="xs:string($enschedeWFS)"/>
                    </Reference>
                    <xsl:for-each select="$var2_ProcessDescriptions/*:
                    ProcessDescription[fn:namespace-uri() eq '']/*:
                    ProcessOutputs[fn:namespace-uri() eq '']/*:
                    Output[fn:namespace-uri() eq '']">
                        <ProcessedResult>
                            <PositionInchain>
                                <xsl:sequence
                    select="$Position1"/>
                            </PositionInchain>
                            <xsl:for-each
                     select="ns0:Identifier">
                                <OutPutIdentifier>
```

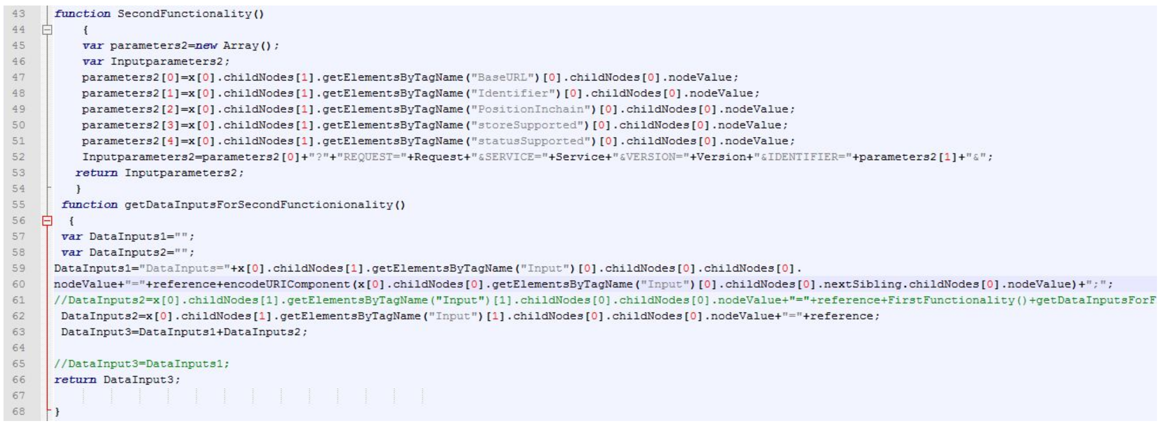
```
                                        <xsl:sequence
                        select="fn:string(.)"/>
                                    </OutPutIdentifier>
                                </xsl:for-each>
                            </ProcessedResult>
                        </xsl:for-each>
                    </Input>
                </xsl:for-each>
            </DataInputs>
        </xsl:for-each>
        <xsl:for-each select="*:ProcessOutputs[fn:namespace-uri() eq '']">
            <ProcessOutputs>
                <xsl:for-each
                select="*:Output[fn:namespace-uri() eq '']">
                    <Output>
                        <xsl:for-each
                select="ns0:Identifier">
                            <Identifier>
                                <xsl:sequence
                select="fn:string(.)"/>
                            </Identifier>
                        </xsl:for-each>
                    </Output>
                </xsl:for-each>
            </ProcessOutputs>
        </xsl:for-each>
    </ProccessingFunctionality>
</xsl:for-each>
<xsl:for-each select="$var1_ProcessDescriptions/*:
ProcessDescription[fn:namespace-uri() eq '']/*:
ProcessOutputs[fn:namespace-uri() eq '']/*:
Output[fn:namespace-uri() eq '']">
    <ChainOutput>
        <PositionInchain>
            <xsl:sequence
              select="$postion2"/>
        </PositionInchain>
        <xsl:for-each
              select="ns0:Identifier">
```

```
                        <OutputIdentifier>
                            <xsl:sequence
                          select="fn:string(.)"/>
                        </OutputIdentifier>
                    </xsl:for-each>
                    <ChainOutputIdentifier>
                        <xsl:sequence select="$FinalOutput"/>
                    </ChainOutputIdentifier>
                </ChainOutput>
            </xsl:for-each>
        </wpschain>
    </xsl:template>
</xsl:stylesheet>
```



```
<html>
<head>
<h1> chaining elemets extarction</h1>
<script type="text/javascript">
var Request="Execute";
var Version="1.0.0";
var Service="WPS";
var reference="Reference@xlink:href=";
var UOM="UOM@=";
if (window.XMLHttpRequest){
xmlhttp=new XMLHttpRequest();
}
else
{
xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}
xmlhttp.open("GET","http://localhost/mywpschain/BufferIntersectionchained2.xml",true);
xmlhttp.send();
xmlDoc=xmlhttp.responseXML;
x=xmlDoc.getElementsByTagName("wpschain");
function FirstFunctionality(){
  var parameters_1=new Array();
  var Inputparameters1;
  parameters_1[0]=x[0].childNodes[0].getElementsByTagName("BaseURL")[0].childNodes[0].nodeValue;
  parameters_1[1]=x[0].childNodes[0].getElementsByTagName("Identifier")[0].childNodes[0].nodeValue;
  parameters_1[2]=x[0].childNodes[0].getElementsByTagName("PositionInchain")[0].childNodes[0].nodeValue;
  parameters_1[3]=x[0].childNodes[0].getElementsByTagName("storeSupported")[0].childNodes[0].nodeValue;
  parameters_1[4]=x[0].childNodes[0].getElementsByTagName("statusSupported")[0].childNodes[0].nodeValue;
  Inputparameters1=parameters_1[0]+"?"+"REQUEST="+Request+"&SERVICE="+Service+"&VERSION="+Version+"&IDENTIFIER="+parameters_1[1]+"&";
return (Inputparameters1);
    }
```

Figure 2: Java script for two proceessing-fucntionalities

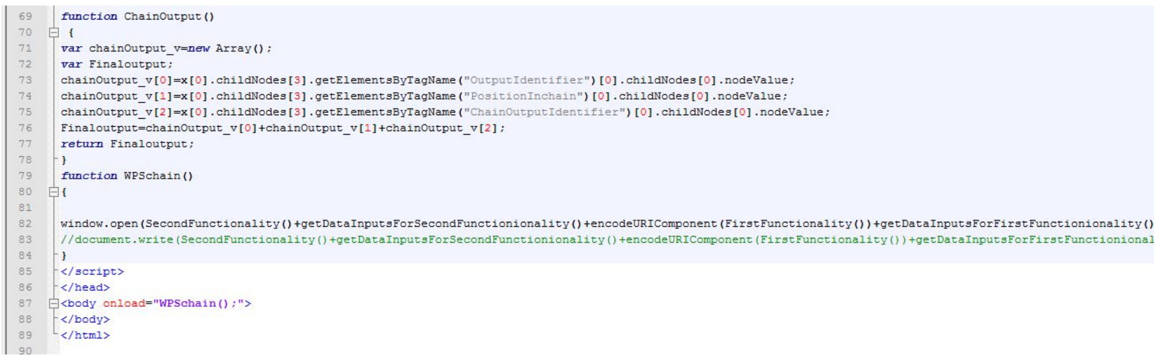


```
function SecondFunctionality()
    {
    var parameters2=new Array();
    var Inputparameters2;
    parameters2[0]=x[0].childNodes[1].getElementsByTagName("BaseURL")[0].childNodes[0].nodeValue;
    parameters2[1]=x[0].childNodes[1].getElementsByTagName("Identifier")[0].childNodes[0].nodeValue;
    parameters2[2]=x[0].childNodes[1].getElementsByTagName("PositionInchain")[0].childNodes[0].nodeValue;
    parameters2[3]=x[0].childNodes[1].getElementsByTagName("storeSupported")[0].childNodes[0].nodeValue;
    parameters2[4]=x[0].childNodes[1].getElementsByTagName("statusSupported")[0].childNodes[0].nodeValue;
    Inputparameters2=parameters2[0]+"?"+"REQUEST="+Request+"&SERVICE="+Service+"&VERSION="+Version+"&IDENTIFIER="+parameters2[1]+"&";
   return Inputparameters2;
   }
 function getDataInputsForSecondFunctionionality()
  {
 var DataInputs1="";
 var DataInputs2="";
DataInputs1="DataInputs="+x[0].childNodes[1].getElementsByTagName("Input")[0].childNodes[0].childNodes[0].
nodeValue+"="+reference+encodeURIComponent(x[0].childNodes[0].getElementsByTagName("Input")[0].childNodes[0].nextSibling.childNodes[0].nodeValue)+";";
//DataInputs2=x[0].childNodes[1].getElementsByTagName("Input")[1].childNodes[0].childNodes[0].nodeValue+"="+reference+FirstFunctionality()+getDataInputsForF
 DataInputs2=x[0].childNodes[1].getElementsByTagName("Input")[1].childNodes[0].childNodes[0].nodeValue+"="+reference;
 DataInput3=DataInputs1+DataInputs2;

//DataInput3=DataInputs1;
return DataInput3;

}
```

Figure 3: Java script for two proceessing-fucntionalities continued from 2



```
function ChainOutput()
 {
var chainOutput_v=new Array();
var Finaloutput;
chainOutput_v[0]=x[0].childNodes[3].getElementsByTagName("OutputIdentifier")[0].childNodes[0].nodeValue;
chainOutput_v[1]=x[0].childNodes[3].getElementsByTagName("PositionInchain")[0].childNodes[0].nodeValue;
chainOutput_v[2]=x[0].childNodes[3].getElementsByTagName("ChainOutputIdentifier")[0].childNodes[0].nodeValue;
Finaloutput=chainOutput_v[0]+chainOutput_v[1]+chainOutput_v[2];
return Finaloutput;
}
function WPSchain()
{

window.open(SecondFunctionality()+getDataInputsForSecondFunctionionality()+encodeURIComponent(FirstFunctionality())+getDataInputsForFirstFunctionionality()
//document.write(SecondFunctionality()+getDataInputsForSecondFunctionionality()+encodeURIComponent(FirstFunctionality())+getDataInputsForFirstFunctional
}
</script>
</head>
<body onload="WPSchain();">
</body>
</html>
```

Figure 4: Java script for two proceessing-fucntionalities continued from 3