GIMA

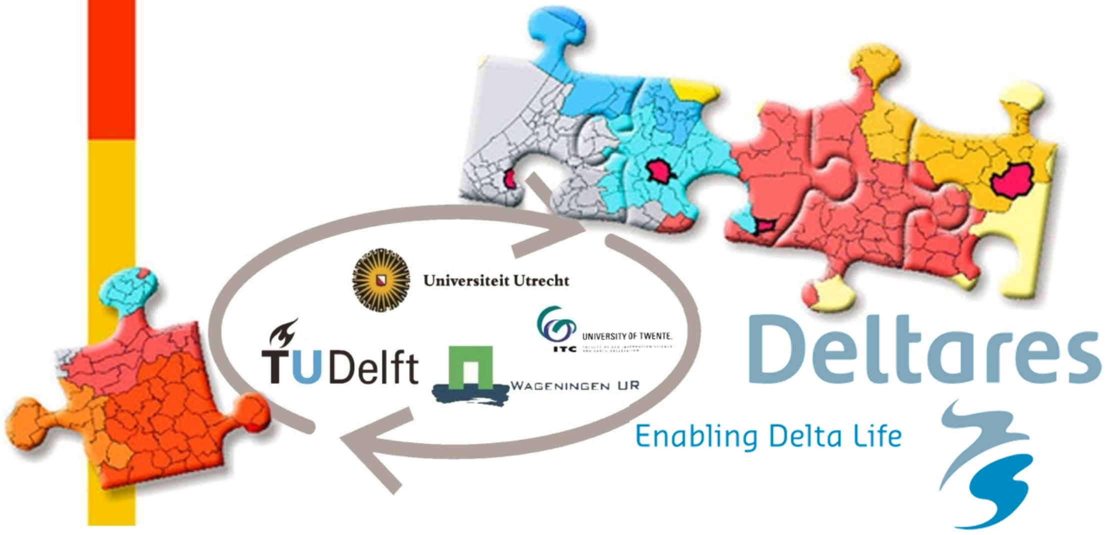
Geographical Information Management and Applications

Implementing the WPS Standard

-

A Case Study for Dissemination of Coastal and Marine Tools

Student:	Joost Boerboom
Supervisors:	Fedor Baart
	Gerben de Boer
	Marian de Vries
Professor:	Peter van Oosterom
Date:	July 16, 2013



Implementing the WPS Standard

A Case Study for Dissemination of Coastal and Marine Tools

MSc GIMA Thesis

Contact Information

Student:	ing. J. Boerboom
ITC#:	29300
UU#:	3824721
Email:	joostboerboom@gmail.com
Phone:	+31 6 41921002
Address:	Zuidpolderplein 12
	2032NV Haarlem
	The Netherlands

Supervisors

Professor:	prof.dr.ir. P.J.M. van Oosterom (Peter)
Deltares:	dr. F. Baart (Fedor)
	dr.ir. G.J. de Boer (Gerben)
TUDelft:	drs. M.E. de Vries (Marian)

Preface

This thesis completes my Master program Geographical Information Management and Applications (GIMA) at Delft University of Technology, University of Twente, University of Utrecht and Wageningen University. In this project I was not only involved in the Web Processing Service, but also in different topics concerning coastal sciences which on their turn have strong relationships to Geographical Sciences. The project really showed me how much Geographical Information is involved in almost anything and how it can be fundamental for many other fields of science.

The thesis work was carried out at Deltares in Delft. After completing my internship at the Hydraulic Engineering Department, Deltares offered me a position as Thesis Student at the Deltares Software Centre. My first gratitude goes to Deltares for offering me the opportunity to work on my thesis in a great atmosphere.

A special thank you goes to Fedor Baart and Gerben de Boer, my daily supervisors at Deltares. Fedor, thank you for helping me out with my Python scripts and your amazing ability of working with all kinds of software and systems required for the project. Gerben, thank you for sharing your knowledge on tidal processes and making Web Services feel as simple as going to the candy shop.

I also want to thank professor Peter van Oosterom and my supervisor Marian de Vries from Delft University of Technology. Thank you for your enthusiasm about the topic and critical reflections on my report.

A final thanks to all others supporting me in any way with completing my Master, my girlfriend, family, friends and all other students and employees working at Deltares.

Joost Boerboom
Delft, July 2013

Summary

The Open Geospatial Consortium has been developing a wide range of open standards for the field of Geographical Information (GI). One of the standards allows for remote processing; the Web Processing Service (WPS). WPS has not been implemented in many project yet but it is a promising standard which can also be applied outside the field of GI.

In this project the WPS standard is applied by means of two case studies, the first is on tidal forecasting and the second is on wrapping an existing remote processing service. The focus of the project is not on making the most optimal and reliable tools but on demonstrating the possibilities WPS offers for a wide range of applications and interoperability between different systems. The goal of the research is to expose complex processes that feed marine and coastal models via a WPS.

To allow for practical testing with the WPS standard a server system was set up. The used server implementation is PyWPS. PyWPS is a Python based open source system which allows execution of Python code directly in WPS processes. During a period of testing PyWPS was found to be running stable on both a Windows computer and Linux web server.

The tidal forecasting service was specifically developed for the project and uses the TOPEX/POSEIDON constituent dataset from Oregon State University. The service allows users to request the water level at any place and time as a result of the astronomical tide. The implemented service offers different output types such as simple graphs and tables which could be used as input for (extensive) models.

The second case implements a WPS based wrapper for executing an already existing Matlab based processing service. This wrapper service offers limited functionality compared to the existing service but is intended for demonstrational use to show that also legacy services can be implemented as WPS.

Both cases are implemented successfully, with an open path for improvements. Both services are not optimized for fast processing. The tidal forecasting service can be improved by making improvements in the script for process optimization, also the dataset can be loaded in a more efficient way for example by tiling or caching. It is expected that the wrapper service cannot be improved very easily. The main issue is that the original tool is relatively slow and WPS is not able to change that unless the original service is improved.

Based on the experiences it is concluded that the WPS standard offers good opportunities for remote processing services in many fields of work. The standard however can be improved in some ways: there should be an option to stop a running process; dynamic status updates should keep the client informed about the running process; in the Describe Process output

there should be an option to communicate complex data such as a model extend. Besides these recommendations for updating the standard it is advised to make processes more dynamically by chaining or coupling with the possibility to change or add parameters after partial runs. Depending on the transfer time of data also process shipping should be considered. In that case not datasets have to be send to a processing server.

For continuing development and improvement of the case studies the following recommendations are made. In the tidal forecasting tool it is advised to implement the latest version of the TPXO dataset which offers better resolution and is validated with actual measurement stations. Also the script should be optimized to reduce processing time.

The wrapper service now offers a limited functionality compared to the original tool, this should be improved to make it in line with the original tool. Sub processes can be added by chaining or coupling these processes. The service is limited on status support, communicating statuses to the user can be important, especially in long running processes.

For both case studies a more practical interface should implemented. Currently only coordinates can be given as input. It is preferred to allow for other input types as well; these types are for example vectors, features or even files of different types.

Overall it can be concluded that WPS offers great options for interoperability of different systems for many fields of science. Adoption of the WPS standard in desktop GIS software allows for better integration of coastal and marine models with GI analysis tools. Still there is room for improvement and extension of the abilities of the standard, however some have already been proposed in the change request for a future version of WPS.

Table of Contents

1. Introduction 1
  1.1 Remote Processing and Cloud Computing 1
  1.2 Web Processing Service Standard 3
  1.3 Goal definition 4
  1.4 Research Questions 5
  1.5 Scope 5
  1.6 Methodology 6
    1.6.1 Literature Study 6
    1.6.2 Test Cases 7
    1.6.3 Relevance 7
2. WPS Background 9
  2.1 Web Services 9
  2.2 Web Processing Services in Detail 10
    2.2.1 WPS Request Build Up 10
    2.2.2 WPS Output Response 15
  2.3 Evaluating the WPS Standard 15
    2.3.1 Services Under Development 16
    2.3.2 As Data Source 16
    2.3.3 Need of Processing Power 16
    2.3.4 Suggested Enhancements 17
  2.4 Process Coupling 18
  2.5 Projects Using WPS 19
    2.5.1 Models as Web Services 19
    2.5.2 Geostatistics via WPS 20
    2.5.3 Commercial Interest 21
  2.6 Other Methods for Remote Processing 21
3. WPS System Setup 23
  3.1 Server 23
  3.2 WPS Clients 23
  3.3 WPS Server Implementations 24
    3.3.1 52North 25
    3.3.2 PyWPS 25
    3.3.3 Choosing a Suitable WPS Server 26
4. Case 1 - Tide Forecasting Service 27
  4.1 Tides: Measurement and Analysis 27
  4.2 Remote Sensing of Water Levels 29
  4.3 Forecasting the Tidal Wave 30

4.3.1 TPXO7.2 Dataset 32
4.4 Tide Forecasting Service 33
4.4.1 Constituent Look-Up tool 33
4.4.2 Basic Forecasting Tool 34
4.4.3 More Advanced Functionalities 37
5. Case 2 – Interactive Dredge Planning Tool 41
5.1 Purpose of IDPT 41
5.2 Using IDPT as Original Service 41
5.3 Method for Wrapping IDPT 43
5.4 Wrapping IDPT as WPS 45
6. Discussion 49
6.1 PyWPS 49
6.2 Test Cases 49
6.2.1 Issues Found in the Forecasting Service 50
6.2.2 Issues Found in Wrapping IDPT 50
6.2.3 QGIS integration 51
6.2.4 Performance 52
7. Conclusions and Recommendations 53
7.1 Conclusions 53
7.2 Recommendations 56
7.2.1 Tide Forecasting Tool 56
7.2.2 IDPT Wrapper 57
7.2.3 Other Recommendations 57
8. References 59
9. Appendices 61
I. Setting up PyWPS in a Windows environment 63
II. Implementation of WPS Processes 65

1. Introduction

Coastal and marine areas are under constant influence of many physical processes and human actions. Especially in areas where sedimentation and deposition speeds are high (soft-shore areas) the underwater topography can change very rapidly.

In the field of Geographical Information (GI) onshore data can still be quite accurate a year (or longer) after collection. In dynamic offshore areas however, not only is data collection harder and more expensive compared to onshore collection events, also the temporal factor is of great importance. Over short periods of time dynamic areas can change in a period of days or even hours.

Instead of surveying large coastal or marine areas, a lot of physical processes are modelled for predicting future changes and influences of certain processes. Besides the fact that offshore surveys are very expensive, it is often not even possible to survey all specific processes at all locations. This can be for example due to the remoteness of the location or just the lack of suitable technologies for measuring.

The modelled processes are often used as an input for a more extensive computational model. Such models are for example able to investigate coastal safety or morphological dynamics, not only do these models interpolate in space but also in time to make predictions. For fast and easy set up of extensive models it is important that all input data are findable and available. Besides the availability, data should also be of a suitable format for use in specific modelling environments, data but also models should be interoperable.

1.1 Remote Processing and Cloud Computing

Extensive models often require heavy computational power and large amounts of input data. On a regular computer for certain models it is quite common to have runtimes of multiple days for relatively small areas. The input data for the models are complex and multi-parameter, depending on the kind of model input data can for example be bathymetry, wave climate, tidal climate, wind conditions and many others.

Many of such processes are modelled by different institutes and often datasets are publicly available. In those situations the actual model processing has already been performed but more processing possibilities can be thought of in this case. One may for example only need a certain area of the total model extend. In that case, a simple tool could be used to clip that area from the result which reduces time for data transfer.

Another option for processing is combining different model results. Deltares is interested in modelling wave heights at the Dutch coast. The required input parameters are derived from different datasets and model outputs. Bathymetries for example are provided by Rijkswaterstaat, wind conditions by the KNMI and tidal levels are derived from a 'local' database at Deltares.

These tidal levels are usually specifically calculated for certain areas which can be relatively time consuming. An interesting possibility would be to offer such calculations as a remote service. The user could then be able to specify his preferred locations and periods where ever the data is needed. Standardizing such services simplifies the modelling operations since the model can now request the data from the service which calculates the required parameters (at the specific location and period) on the fly.

Remote processing can also offer a solution for remote areas where internet access and computational power is limited. To reduce the amount of data being transferred and limit the amount of computational power, (many) models can be run on a remote computer or server. This technology does not solely give a solution for these remote areas but also for the use of 'thin clients' and mobile devices such as smart phones and tablets. By using a remotely processed model, the user for example only has to provide input parameters, the computer on which the model is actually running gathers the required input datasets and runs the model. After finishing the model run, the user receives the output for example as an image, map or dataset.

Another name for remote processing is cloud computing. We are moving lots of data to the 'cloud' by using so called 'cloud' storage like Dropbox and Google Drive. What the cloud actually is, is not defined sharply. Weiss (2007) refers to actual clouds which can have about any shape and size. Usually with the cloud the entire internet is meant, besides storage the internet or the cloud is also able to run processes. Running processes in the cloud is usually referred to as distributed computing or cloud computing.

In cloud computing applications are moved from the user's computer to the cloud. In the cloud software is offered as service. Well known services are for example email or online text editors like Google Docs. Other services can also be a Desktop Service, in a Desktop Service a user can for example access the application via a website where it can be used like it was on his local system. Several companies make use of a Citrix environment. In this environment a 'regular' operating system is run on a server for each user. Employees can connect to the server and work via the connection like they work from a local operating system. The difference is that the local computer is only used to send input operations (keystrokes, mouse movements, etc.) and receive visual output.

Intermezzo: Web Services

People working with datasets (from different providers) are often used to downloading files from many sources. Whenever a new version of a dataset is published, the data should be downloaded again, when updating these local data is neglected results are outdated even before publishing. The use of Web Services can make a change in these imperfections.

To find out what Web Services actually are we make take a quick look on the internet and find a description which states that a web service is the representation of a communication method over the internet. Another method for internet communication that may come up is email. Using an online accessible mailbox offers the user to send and receive messages (data) over the internet without storing anything on your own computer.

In the working field of GI multiple standards for Web Services are available. The best known Web Services are able to handle data or processes, services that are capable of handling different types are available as well. A service for data is probably the easiest one to get a view on. Just like with webmail, the data Web Service (usually) offers multiple datasets (messages) which can be accessed by the client. The clients sends an instruction (request) to the service which responds by means of a dataset which can be for example a (piece of a) map.

Web Services do not solely allow clients to access data, but also to use data. Data from Web Services can be directly used in for example Desktop GIS's or directly as input for online processing Web Services. Web Services offer quite some advantages compared to 'old fashioned downloading' of datasets. Using a Web Services ensures the client to have the latest version and can be accessed from any device at any time. Also sharing of data is easy, instead of sending (big) files to each other, just a simple link to the Web Service will do the job.

1.2 Web Processing Service Standard

The Open Geospatial Consortium (OGC) is an international organisation which focuses on the development of open standards for GI. From the start in 1994 the OGC has developed 39 standards (February 2013) to improve interoperability between different fields of science (OGC, 2013).

A specific range in the OGC standards is the 'Web Service' (WxS) standards family. These standards apply for ways of disseminating GI data and services over the internet. One of these 'Web Services' is the 'Web Processing Service' (WPS). WPS is a communication standard which allows users to access tools or models which can be applied on geographical data.

Accessing a WPS can for example be done via a website, in that case there is often some other web service (for example a 'Web Map Service' (WMS)) to provide the input parameters such as location of the modelling area. Another option for accessing a WPS is by using a desktop GIS software package. Since the WPS is an open standard, anyone is allowed to implement and use the standard. Currently, most well known GIS software packages (e.g. ArcGIS and Q-GIS) have implemented a WPS client. An example of a WPS set up can be found in Figure 1.1.

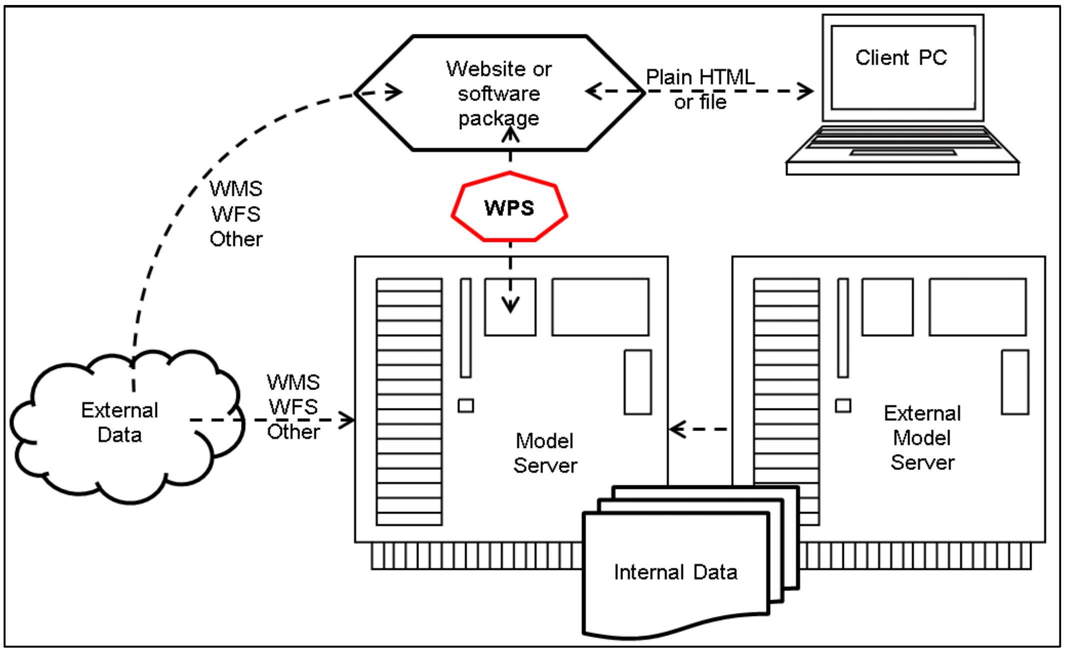


Figure 1.1 - Example of a WPS setup.

A WPS is built from different pre-programmed processes. These processes can be called and executed on a specified dataset. For such executions the command is actually included in the URL which is sent to the server.

A process can for example be a simple operation on the data such as a coordinate conversion or buffer creation. In a process it is also possible to, for example, run a program on the server. In this case a model on the server can be executed by the WPS. It is also possible to couple different processes, this is also known as process chaining.

1.3 Goal definition

The goal of this thesis is to *"expose complex processes that feed coastal and marine models via a Web Processing Service"*. To reach this goal, a few sub goals are defined. Below these sub goals are described and explained.

1. Set up a (simple) Web Processing Service suitable for testing different operations.

This sub goal aims at finding suitable methods for setting up a Web Processing Service which can later be used for dissemination of other processes from the following goals, two and three. The service set up is used as proof of concept of the WPS standard and available software. To reach this goal example processes are used to find out the correct syntax for implementing ‘regular’ scripts as WPS scripts. The result of this goal an operational WPS server.

2. Implement a tide forecasting tool which can be disseminated as a Web Processing Service for data submission, for example for a modelling environment.

This sub goal on itself contains two activities. The first activity focuses on finding a suitable method for predicting the water levels as a result of astronomical influences. This method should be able to use remote sensing (satellite) data as input. The second activity is implementing this forecasting method in a new WPS process. It should also be possible to disseminate the tool outside Deltares so it could be useful for a wide audience. For that reason there should be no restrictions to for example re-use of existing code or datasets. The second activity is to implement this method of forecasting in a Python script which can be used as a WPS.

3. Implement an existing model as a Web Processing Service.

This goal focuses on controlling an existing model by means of WPS. Such a WPS compliant service is also known as a wrapper. In this case no activities are planned on the real content of the service but on the method of controlling the service. The reason why such a wrapper is interesting to build is that this can show the option of disseminating existing models and services in a standardized way. Several models and tools are available (at Deltares) beforehand but these lack the kind of interoperability that the WPS standard offers.

1.4 Research Questions

Besides the goals set in the previous paragraph, a number of research questions are defined below.

1. Can complex processes that feed coastal/marine models be exposed via a Web Processing Service?
2. Can a Web Processing Service be used as a wrapper service for non domain-specific processing services?
3. Which adaptations could be made to the WPS standard to improve the possibilities for application of the standard?

1.5 Scope

Applicability and research on WPS can be very wide spread. The focus of the project is on the application of the WPS standard on tools used in coastal and marine engineering.

In the literature study an overview is given of previous research on WPS. Since the amount of publications on WPS, is limited a wide range of topics are reviewed. Besides the academic literature, also the standard documents are studied. As a preparation for the implementation, also documentation and tests of different software packages are studied.

The practical work exists of setting up a WPS server and programming two services. For each service some theory is introduced which is required for understanding the working and the purposes of the services.

The first function, (section 4) Tide Forecasting Service is used as a start up for working with WPS's. Different versions of the service are made which use the same Python function as starting point. This part is the function that calculates the water level with the data from the TPXO dataset. Versions based on this starting point mainly differ in output type (e.g. graph or text) and are used to explore the possibilities of WPS.

The second function, (section 5) the wrapper of the Interactive Dredge Planning Tool is used to demonstrate the possibilities of the WPS standard in standardizing the operations of already existing services. The WPS service will be limited compared to the original tool, although it is fully operational, different parameters can be given as input and the output gives a file actually produced by the original service based in the input parameters provided by the WPS wrapper. An important part of this service is providing statuses in the operation sequence. These statuses are used for communication between the different services and clients. The status of a process can for example contain information about the expected run time or provide a chained (following) process with the location of the output data.

A graphical like interface for WPS services is not part of this project. The services resulting of this project can all be called from a web browser. Although an interface can offer more convenient operations, for the actual goal of the project such an interface is not very relevant. For future use of WPS Deltares is internally working on a WPS interface in their own modelling environment.

1.6 Methodology

The thesis work contains different methods of working. Starting the thesis will mainly exist of a literature study on previous implementations of the WPS standard and a study on how to set up a WPS based service. During the literature study existing WPS implementations are studied as well in practise. In that case it is tried to find out the working of such a WPS. Two test cases are set up to figure out to what extend relevant processes can be disseminated using the WPS standard.

1.6.1 Literature Study

A literature study was performed on different topics involved in the project. Topics such as web services, the WPS standard and tidal forecasting pass in review. Information on these topics is preferred to be received from scientific papers or (academic) textbooks. Other options for information sources are 'standard documents' and Deltares' experts on for example tidal forecasting and modelling.

Since WPS is quite a recent standard (2007) publications are relatively new as well. The WPS standard is not yet widely applied so most researches will, like this project, focus on application of the WPS standard for a certain field of work. Papers describing specific operations or testing of new functions within the WPS standard can be taken into account for recommendations for further research.

1.6.2 Test Cases

In two test cases the application of the WPS standard is demonstrated. Comparable to other researches, services were implemented and disseminated. Development of services will be done according to the DTAP (Development, Testing, Acceptance and Production) structure. In the procedure the Production phase is excluded.

With the test cases some additional operations are planned. At first a suitable WPS server implementation was found. A server instance gives the service provider the opportunity to offer WPS based services in a relative easy way. For most instances little changes to basic scripts have to be made. Requested specifications for the server instances are the option to use the Python programming language offer smoothly operability.

Programming (*D – Development*) of the services will be done in the Python language. Python is chosen because of previous experience with the language, popularity in the GI community and being an open language. The first phase of programming consists of creating the actual tools without the implementation of the WPS standard. In this phase the IPython Notebook editor will be used, this editor offers a clear overview all code blocks and keeps partial outputs visible.

The final phase will be dedicated to implementing the chosen processes as WPS service. The created Python code from the first programming phase is now integrated in the script layout suitable for the chosen server instance. At completion the WPS script will be uploaded to the server where it can be tested (*T – Testing*). After the testing phase it is expected that some changes have to be made before an operational version of the service is launched (*A – Acceptance*).

1.6.3 Relevance

The first test case will be a service for forecasting water levels according to the astronomical tide. Tide is an often used parameter for coastal modelling and flood risk assessments, both in line with Deltares' field of work. Also, tide forecasting services can be relevant for many more applications, not only professionally but also for example for recreational use such as scuba diving or surfing. Considering this, an implemented tide forecasting service, which provides forecasted water levels independent of the location (global model), may gain quite some popularity among different user communities.

The second test case is an existing Deltares service, the 'Interactive Dredge Planning Tool' (IDPT). This tool offers an interface where users can plan dredging operations in the

Singapore area (model extend). Based on several parameters the tool calculates the sediment plume and indicates the environmental impact of this plume (Ecoshape, 2012). IDPT already runs as a remote processing service but not conform the WPS standard. In this project a so called wrapper is made which allows the running of the tool via a WPS service. The functionality offered by this wrapper is limited in the sense of input parameters compared to the actual tool. Since this is only made for demonstrational use, no issues are expected with this, also the modelling results are similar (with use of some fixed parameters) to the non-wrapped tool.

2. WPS Background

The WPS standard has been defined by the OGC in 2007 (OGC et al., 2007). In the period up to today only a (relatively) limited number of WPS implementations have been published. From a total number of 9329 harvested web services (WxS's) based on the OGC standards in 2011, only 58 were based on WPS (Lopez-Pellicer et al., 2012). In his research Lopez-Pellicer used a 'crawler' to discover Geographical Web Services which applied to the OGC standard. This crawler however could not search for services registered in catalogues or services behind commercial borders and firewalls.

For several reasons many WPS's are not publicly available. It does take some experience to use certain services and besides, it requires computational power from the service provider. However, like everywhere on the internet, several free services or software packages (open source) are available for fair use. Also the open data policies of many governments may stimulate to provide for some (simple) processing service in their portals. Using certain combinations of datasets, processes (tools) and input parameters can actually be interesting for different parties. When a service provider could archive all input and output of processing proceedings these could be used to get an insight in the way people like using their tools and datasets and make improvements on their won systems.

In literature (e.g. Michael & Ames, 2007; Zhao et al., 2012; Castranova et al., 2013) different topics concerning the WPS standard are researched. In the following sections these topics are highlighted from review of the standard itself up to application of the standard in an operational service. Since the standard is relatively new, most available papers are quite recent. A disadvantage of reviewing recent developments is that the amount of publications available is somewhat limited.

This chapter gives an overview of some different aspects concerning WPS. Section 2.1 informs about Web Services in general, 2.2 explains the way of communication (requests) with WPS. Sections 2.3, 2.4 and 2.5 discuss what is found in literature and 2.6 gives some methods different from WPS for remote processing.

2.1 Web Services

A Web Service is: "A software system designed to support interoperable machine-to-machine interaction over a network" (W3C, 2004). In general these services run on web servers and can be accessed from a client's PC with a web browser or a specific piece of client software (Vaughan, 2002).

Web Services have been evolved from the Remote Procedure Call (RPC), a software development framework origination from 1990 (Kalin, 2009). The name Web Services was first introduced by Microsoft in the year 2000 (Weaver, 2004). The OGC started the initiative for open web services in 2003, the first standard was launched in 2004, the Web Mapping Service (WMS) (OGC, 2013).

Michael and Ames (2007) describe some other initiatives for geographical web services that came from software vendors. ESRI for example launched the Geoprocessing Server in 2003 (ArcInfo 8.3) which could process jobs submitted from clients. However this service was only available for ESRI clients and used a closed protocol. The service became unavailable in the next version (ArcGIS 9.0). The reason for removing such remote processing functionality is unknown, an obvious reason could be because of limited profits generated by the service.

Currently most desktop GIS software packages support OGC Web Service standards and are in that way able to handle WPS requests as a client. As an example, ArcGIS imports a WPS server as a toolbox so services can be used stand alone or in het model builder. This last option simplifies chaining of multiple processes, even from different servers.

The most important standard for data exchange is currently XML (Extensible Markup Language), a W3C standard, and is a base for other Web Service standards (Vaughan-Nichols, 2002). XML is used for representing structured information of about anything. The advantage of using XML is that it is self descriptive. Users can choose their own tags which can make it relatively simple to understand the format of a file but also to find errors (W3C, 2010).

Web Services used for geographical applications often use GML (Geography Markup Language) as in- and output for datasets. As stated by de Vries and van Oosterom (2008), GML is especially designed for geographical (point, vector and polygon) data and is able to hold complex geometry and topology data. For encoding spatial properties of geographical objects the GML core contains many geographical data types. Besides that, the GML standard specifies how other domain specific spatial and also non-spatial data types are derived. GML allows the encoding of topology in the XML (standard) syntax.

2.2 Web Processing Services in Detail

The following section describes the actual way of performing operations using WPS. Different input requests and a general WPS output file are discussed.

2.2.1 WPS Request Build Up

In its most basic form a WPS is accessed from a web browser and is operated using the Hypertext Transfer Protocol (HTTP). Used methods within HTTP for accessing a WPS are Get and Post. Get is used for retrieving data or information, for example retrieving data from a server. The Post method is used to deploy data from a client to the server, for example uploading a file, posting a message on a forum or sending a request (W3C, 1999).

In the WPS standard three different operations are defined: GetCapabilities, DescribeProcess and Execute. Adding parameter identifiers and values to these operations result in a complete request which can be sent to the WPS server using HTTP Get, in this case the server responds by sending an XML file. Sending the request is as simple as typing a URL (Uniform Resource Locator) in a web browser. A second method is to do a request by using HTTP Post, in this method a XML file is sent from the client to the server. Usually HTTP Get is used for GetCapabilities and DescribeProcess operations; HTTP Post is most used for the Execute operation. How different methods of HTTP are used and how WPS requests are built up is described below.

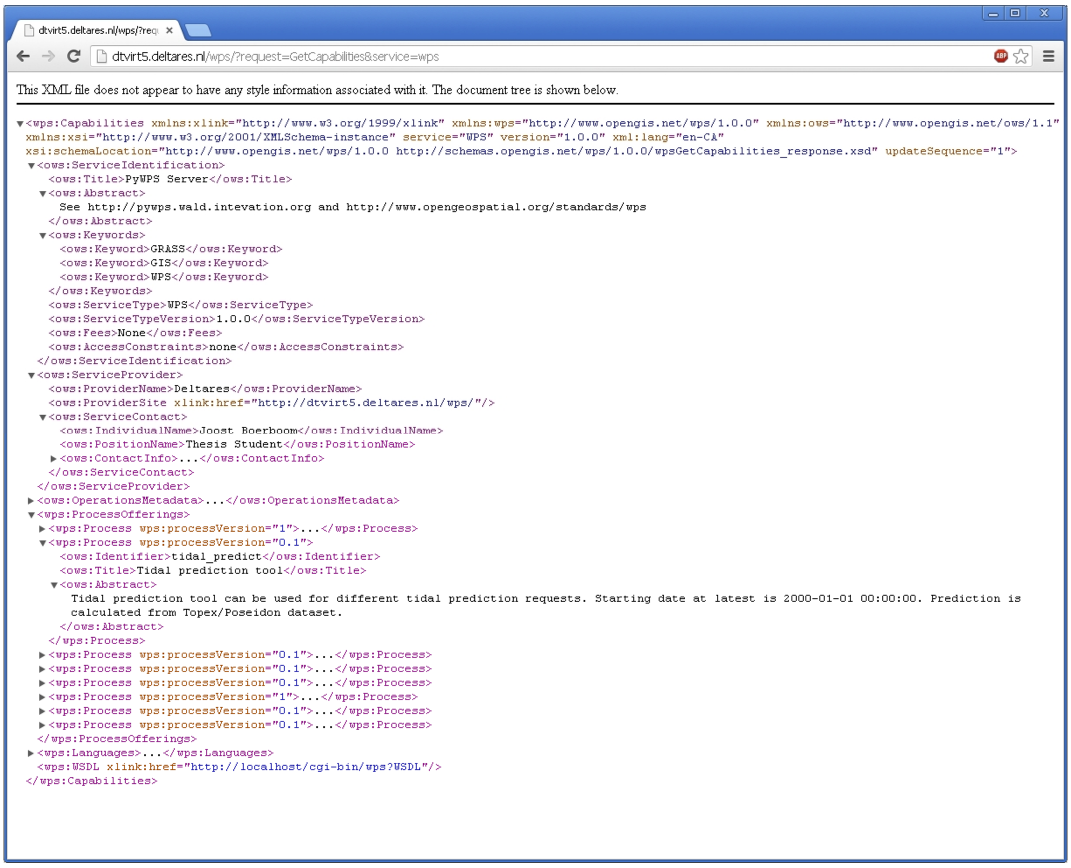
GetCapabilities

In the GetCapabilities request, the server answers with the metadata of the service. This metadata describes the abilities of the requested server. The GetCapabilities operation returns the names and descriptions of the different pre-programmed processes supported by the specific WPS server. The GetCapabilities request is also used in other OGC web services.

The GetCapabilities can be further specified with the ‘AcceptVersions’ and ‘language’ identifiers. These identifiers can for example be used to limit the result to a specific WPS version or language. An example of a complete GetCapabilities request can be found in Figure 2.1. A GetCapabilities output example is in Figure 2.2.

`http://localhost:8080/wps.cgi?`	Address of the server
`    service=WPS&`	Type of Web Service
`    request=GetCapabilities&`	Type of request
`    AcceptVersions=1.0.0&`	Service version limitation
`    language=en`	Language limitation (English)

Figure 2.1 - Example of a GetCapabilities request using HTTP Get.



dtvirt5.deltares.nl/wps/?request=GetCapabilities&service=wps

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wps:Capabilities xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" service="WPS" version="1.0.0" xml:lang="en-CA"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsGetCapabilities_response.xsd" updateSequence="1">
  ▼<ows:ServiceIdentification>
     <ows:Title>PyWPS Server</ows:Title>
    ▼<ows:Abstract>
       See http://pywps.wald.intevation.org and http://www.opengeospatial.org/standards/wps
     </ows:Abstract>
    ▼<ows:Keywords>
       <ows:Keyword>GRASS</ows:Keyword>
       <ows:Keyword>GIS</ows:Keyword>
       <ows:Keyword>WPS</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>Deltares</ows:ProviderName>
     <ows:ProviderSite xlink:href="http://dtvirt5.deltares.nl/wps/"/>
    ▼<ows:ServiceContact>
       <ows:IndividualName>Joost Boerboom</ows:IndividualName>
       <ows:PositionName>Thesis Student</ows:PositionName>
      ▶<ows:ContactInfo>...</ows:ContactInfo>
     </ows:ServiceContact>
   </ows:ServiceProvider>
  ▶<ows:OperationsMetadata>...</ows:OperationsMetadata>
  ▼<wps:ProcessOfferings>
    ▶<wps:Process wps:processVersion="1">...</wps:Process>
    ▼<wps:Process wps:processVersion="0.1">
       <ows:Identifier>tidal_predict</ows:Identifier>
       <ows:Title>Tidal prediction tool</ows:Title>
      ▼<ows:Abstract>
         Tidal prediction tool can be used for different tidal prediction requests. Starting date at latest is 2000-01-01 00:00:00. Prediction is
         calculated from Topex/Poseidon dataset.
       </ows:Abstract>
     </wps:Process>
    ▶<wps:Process wps:processVersion="0.1">...</wps:Process>
    ▶<wps:Process wps:processVersion="0.1">...</wps:Process>
    ▶<wps:Process wps:processVersion="0.1">...</wps:Process>
    ▶<wps:Process wps:processVersion="1">...</wps:Process>
    ▶<wps:Process wps:processVersion="0.1">...</wps:Process>
    ▶<wps:Process wps:processVersion="0.1">...</wps:Process>
   </wps:ProcessOfferings>
  ▶<wps:Languages>...</wps:Languages>
   <wps:WSDL xlink:href="http://localhost/cgi-bin/wps?WSDL"/>
 </wps:Capabilities>
```

Figure 2.2 - Example output of a GetCapabilities request. In the output document information can be found about for example the provider, available services and available languages.

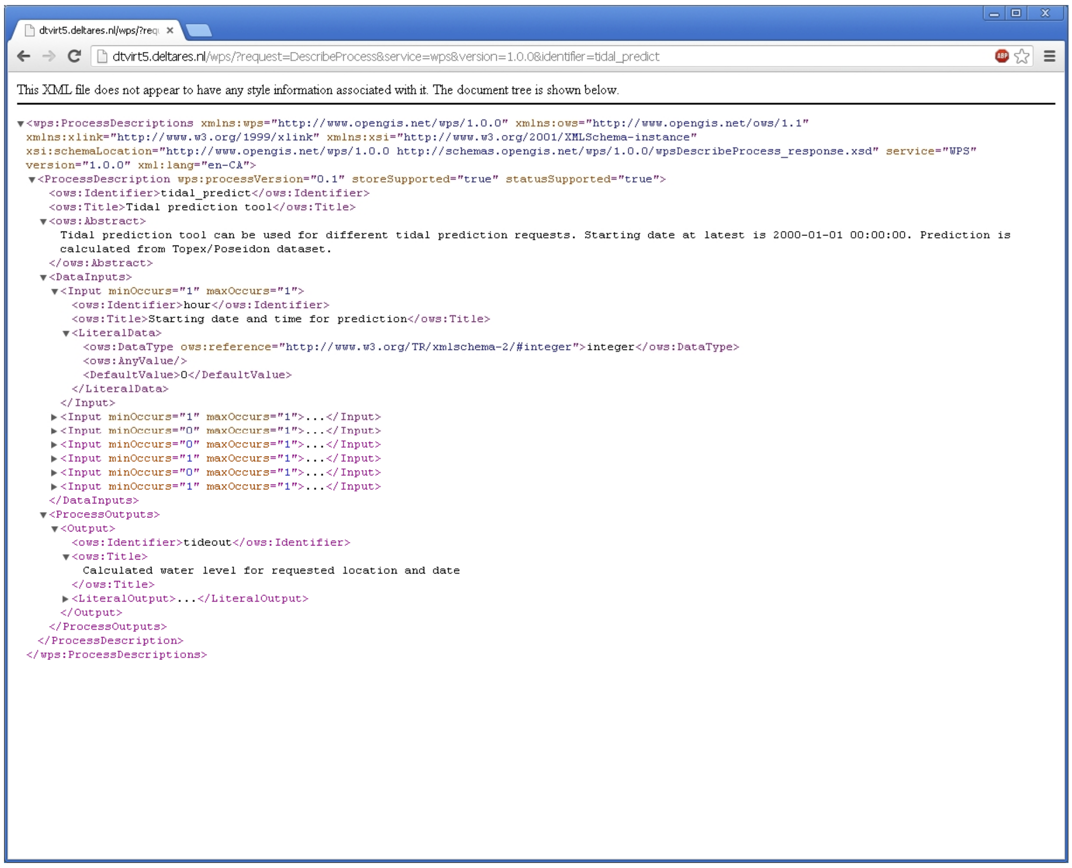
DescribeProcess

The DescribeProcess request provides the client with more detailed information on a certain process that can be run on the WPS server. The information is send to the server which replies with an XML file containing information on the requested process.

Replied information shows all (available) meta-information about the specific process. Besides, all required in- and outputs are specified. There is also the possibility to include a user manual or a reference to a web site. An example of a DescribeProcess request can be found in Figure 2.3. A DescribeProcess output example is in

`http://localhost:8080/wps.cgi?`	Address of the server
`service=WPS&`	Type of Web Service
`request=DescribeProcess&`	Type of request
`AcceptVersions=1.0.0&`	Service version limitation
`language=en&`	Language limitation (English)
`Identifier=process1,process2`	Specify requested process(es)

Figure 2.3 - Example of a DescribeProcess request using HTTP Get.


dtvirt5.deltares.nl/wps/?req
dtvirt5.deltares.nl/wps/?request=DescribeProcess&service=wps&version=1.0.0&identifier=tidal_predict

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd" service="WPS"
 version="1.0.0" xml:lang="en-CA">
  ▼<ProcessDescription wps:processVersion="0.1" storeSupported="true" statusSupported="true">
     <ows:Identifier>tidal_predict</ows:Identifier>
     <ows:Title>Tidal prediction tool</ows:Title>
    ▼<ows:Abstract>
       Tidal prediction tool can be used for different tidal prediction requests. Starting date at latest is 2000-01-01 00:00:00. Prediction is
       calculated from Topex/Poseidon dataset.
     </ows:Abstract>
    ▼<DataInputs>
      ▼<Input minOccurs="1" maxOccurs="1">
         <ows:Identifier>hour</ows:Identifier>
         <ows:Title>Starting date and time for prediction</ows:Title>
        ▼<LiteralData>
           <ows:DataType ows:reference="http://www.w3.org/TR/xmlschema-2/#integer">integer</ows:DataType>
           <ows:AnyValue/>
           <DefaultValue>0</DefaultValue>
         </LiteralData>
       </Input>
      ▶<Input minOccurs="1" maxOccurs="1">...</Input>
      ▶<Input minOccurs="0" maxOccurs="1">...</Input>
      ▶<Input minOccurs="0" maxOccurs="1">...</Input>
      ▶<Input minOccurs="1" maxOccurs="1">...</Input>
      ▶<Input minOccurs="0" maxOccurs="1">...</Input>
      ▶<Input minOccurs="1" maxOccurs="1">...</Input>
     </DataInputs>
    ▼<ProcessOutputs>
      ▼<Output>
         <ows:Identifier>tideout</ows:Identifier>
        ▼<ows:Title>
           Calculated water level for requested location and date
         </ows:Title>
        ▶<LiteralOutput>...</LiteralOutput>
       </Output>
     </ProcessOutputs>
   </ProcessDescription>
 </wps:ProcessDescriptions>
```

Figure 2.4 - Example output of a DescribeProcess request. In the output document information can be found about for example about the goal of the process (abstract), input parameters and output.

Execute

The Execute request usually differs from the GetCapabilities and DescribeProcess in the way that the request is sent to the server. Instead of using HTTP Get, the Execute request often uses HTTP Post (although the request can also be executed using HTTP Get). To use this method to execute a process, an XML file containing input parameters is sent to the server. The file should contain all required information specified in the XML from the DescribeProcess request. Sending XML files for executing WPS processes is usually performed from WPS client software.

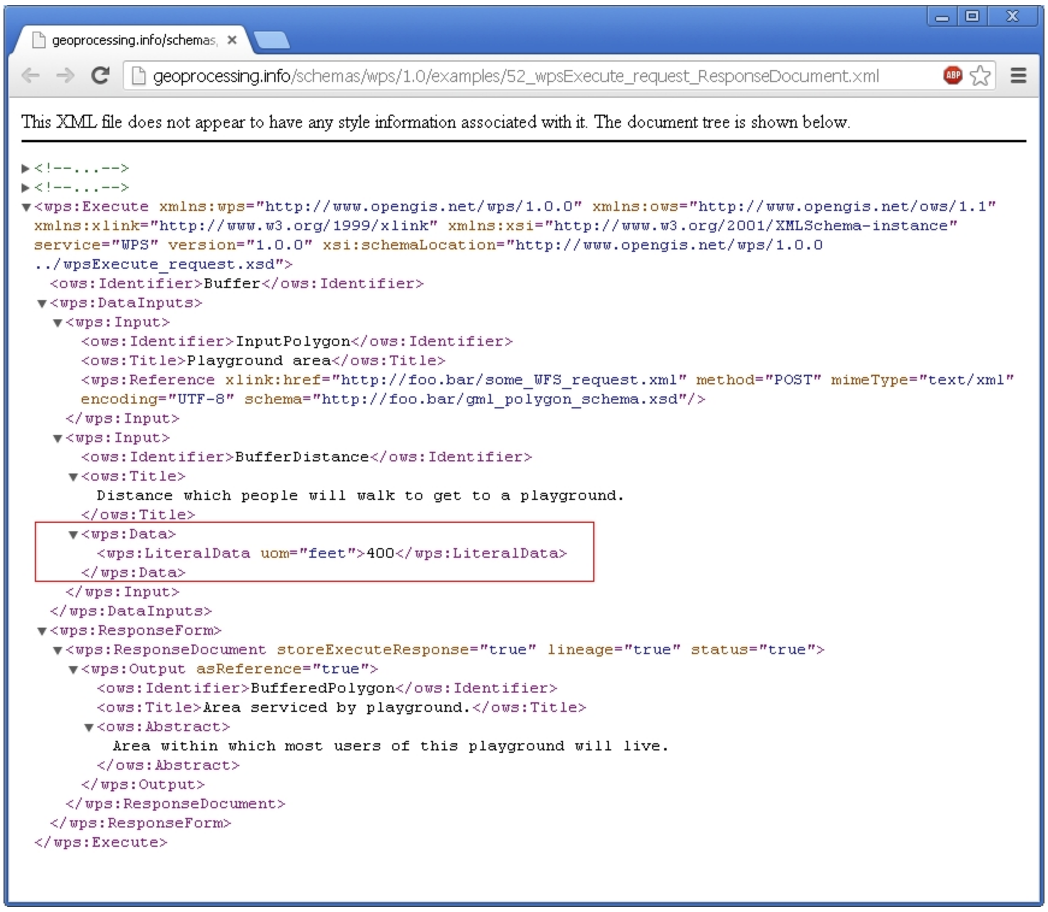
Another option is using HTTP Get for the Execute request. In this case the request is built like the GetCapabilities and DescribeProcess requests. The main addition compared to the previous requests is the DataInputs parameter. In this parameter input values are added and, when needed, data types can be defined by adding '@datatype=' directly after the input value. An example of an Execute request can be found in Figure 2.5.

Using the Execute request with HTTP Post can become very cluttered when a lot of input values and definitions are required for a process. To maintain overview on the request it is advised to use this method only for relatively short requests. For extensive requests the easiest method is to use a request builder (when offered by the service provider) or a (built-in) WPS function in a desktop GIS software package.

```
http://localhost:8080/wps.cgi?         Address of the server
    service=WPS&                       Type of Web Service
    request=Execute&                   Type of request
    AcceptVersions=1.0.0&              Service version limitation
    language=en&                       Language limitation (English)
    Identifier=process1&               Specify requested process
    DataInputs=[input1=x;input2=y]     Input data (values)
```

Figure 2.5 - Example of an Execute request using HTTP Get.

Sending an execute request by means of HTTP Post requires a full XML containing input data. This file is send to the server which uses the document as input for the request. An example of an Execute request XML file is in Figure 2.6


geoprocessing.info/schemas/wps/1.0/examples/52_wpsExecute_request_ResponseDocument.xml

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▶<!--...-->
▶<!--...-->
▼<wps:Execute xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 service="WPS" version="1.0.0" xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
 ../wpsExecute_request.xsd">
   <ows:Identifier>Buffer</ows:Identifier>
 ▼<wps:DataInputs>
   ▼<wps:Input>
      <ows:Identifier>InputPolygon</ows:Identifier>
      <ows:Title>Playground area</ows:Title>
      <wps:Reference xlink:href="http://foo.bar/some_WFS_request.xml" method="POST" mimeType="text/xml"
      encoding="UTF-8" schema="http://foo.bar/gml_polygon_schema.xsd"/>
    </wps:Input>
   ▼<wps:Input>
      <ows:Identifier>BufferDistance</ows:Identifier>
     ▼<ows:Title>
        Distance which people will walk to get to a playground.
      </ows:Title>
     ▼<wps:Data>
        <wps:LiteralData uom="feet">400</wps:LiteralData>
      </wps:Data>
    </wps:Input>
  </wps:DataInputs>
 ▼<wps:ResponseForm>
   ▼<wps:ResponseDocument storeExecuteResponse="true" lineage="true" status="true">
     ▼<wps:Output asReference="true">
        <ows:Identifier>BufferedPolygon</ows:Identifier>
        <ows:Title>Area serviced by playground.</ows:Title>
       ▼<ows:Abstract>
          Area within which most users of this playground will live.
        </ows:Abstract>
      </wps:Output>
    </wps:ResponseDocument>
  </wps:ResponseForm>
</wps:Execute>
```

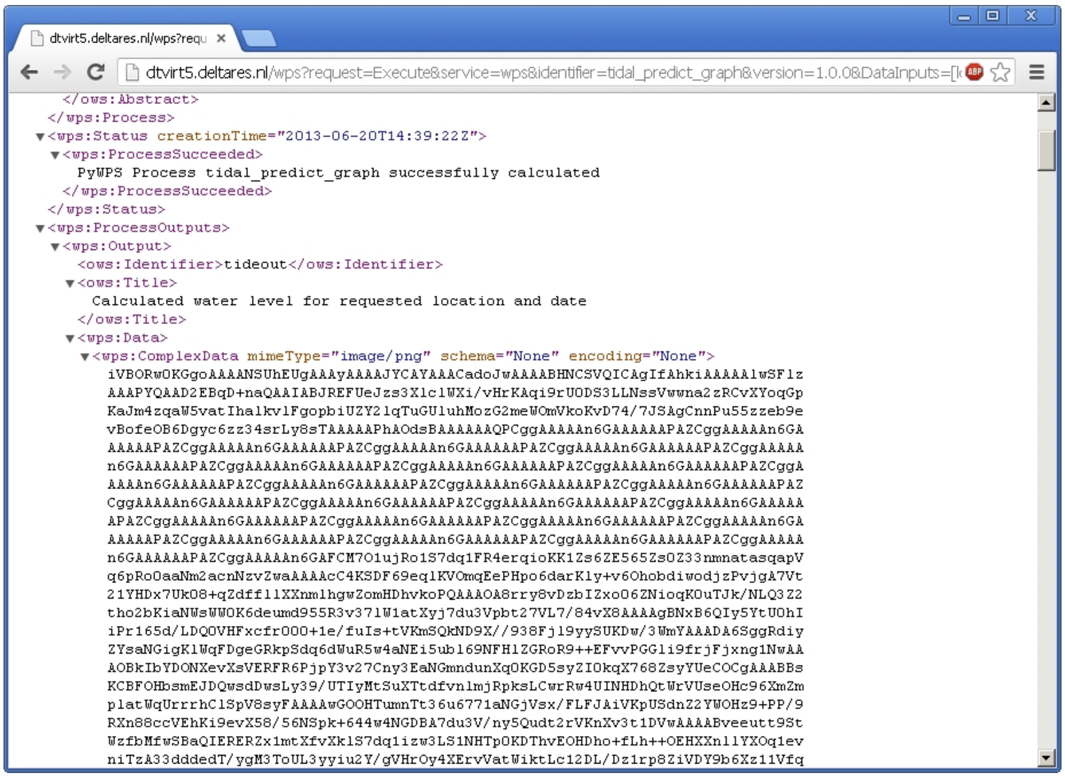
Figure 2.6 - Example[1] of an Execute request XML file which is send to the WPS server by means of HTTP Post. In the red box the input data is defined.

[1] XML file originates from http://geoprocessing.info/schemas/wps/1.0/examples/52_wpsExecute_request_ResponseDocument.xml

2.2.2 WPS Output Response

The output of a WPS process, regardless of the type of request is always an XML document. Like the input requests, also the structure of the output XML is defined in the WPS standard documents. Having a prescribed XML structure makes it relatively easy and attractive for software developers to create applications that can read the WPS XML output documents.

The actual process output is given in between the 'wps:LiteralData' or 'wps:ComplexData' tags. The LiteralData tag refers to a single value (numbers or characters) and the ComplexData refers to a file like output. This can be about any file type, in the output the file type (Mime type) is specified so when the XML document is read by software, the program knows how it should react to the encoded file in the XML. Figure 2.7 gives an example of how a WPS output XML document containing raw data can look like, in this case the raw data represents an image in PNG format.


dtvirt5.deltares.nl/wps?requ
dtvirt5.deltares.nl/wps?request=Execute&service=wps&identifier=tidal_predict_graph&version=1.0.0&DataInputs=[l

```
    </ows:Abstract>
  </wps:Process>
▼<wps:Status creationTime="2013-06-20T14:39:22Z">
  ▼<wps:ProcessSucceeded>
     PyWPS Process tidal_predict_graph successfully calculated
    </wps:ProcessSucceeded>
  </wps:Status>
▼<wps:ProcessOutputs>
  ▼<wps:Output>
      <ows:Identifier>tideout</ows:Identifier>
    ▼<ows:Title>
       Calculated water level for requested location and date
      </ows:Title>
    ▼<wps:Data>
      ▼<wps:ComplexData mimeType="image/png" schema="None" encoding="None">
         iVBORw0KGgoAAAANSUhEUgAAAyAAAAJYCAYAAACadoJwAAAABHNCSVQICAgIfAhkiAAAAAlwSFlz
         AAAPYQAAD2EBqD+naQAAIABJREFUeJzs3Xlc1WXi/vHrKAqi9rUODS3LLNssVwwna2zRCvXYoqGp
         KaJm4zqaW5vatIhalkvlFgopbiUZY2lqTuGUluhMozG2meWOmVkoKvD74/7JSAgCnnPu55zzeb9e
         vBofeOB6Dgyc6zz34srLy8sTAAAAAPhAOdsBAAAAAAQPCggAAAAAn6GAAAAAAn6GAAAAAAPAZCggAAAAAn6GA
         AAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAA
         n6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggA
         AAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZ
         CggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAA
         APAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GA
         AAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAAn6GAAAAAAPAZCggAAAAA
         n6GAAAAAAPAZCggAAAAAn6GAFCM7O1ujRo1S7dq1FR4erqioKK1Zs6ZE565Zs0ZJwv0W+B7zLFN2
         nmnatasqapV
         q6pRo0aaNm2acnNzvZwaAAAAcC4KSDF69eqlKVOqEePHpo6darKly+v6Ohobdiwodj2PvjgA7Vt
         21YHDx7Uk08+qZdff1lXXnmlhgwZomHDhvkoPQAAAOA8rry8vDzbIZxo06ZNKiioqKOuTJk/NLQ3Z2
         tho2bKiaNWsWWOK6deumd955R3v371lWlatXyj7du3Vpbt27VL7/84vX8AAAAgBNxB6QIy5YtU
         iPr165d/LDQ0VHFxcfr000/fuIs+tVKmSQkND9X//938FjI9yySUKDw/3WmYAAAAD6SggRdiy
         ZYsaNGigK1WqFDgeGRkpSdq6c6dWuR5w4aNEi5ubl69NFH1ZGRoR9++EvvPGG1i9frnHjx
         AOBkIbYDONXevXsVERFR6Pi5Cny3EaNGmndndunXq0KGD5syZsyZeqkX768ZsyYUeCOCgAAABBs
         KCBFOHbsmEJDQwsdDwsLy58vSlhYmOxyOXLfQNxLXvGJ2kydfvnlmjRpkskLCwrRw4UINHhDdQtWrVpVUYOKOhc96XmZm
         platWqUrrrhClSpV8syFAAAAwGOOHTumnTt36u6771aNGjVsx/FLFJAiVKpUSdnZ2YWn9+PP/9
         RXn88ccVEhKi9evX58/56NSpk+644w4NGDBA7du3V/ny5VDwAAAABveeutt9St
         WzfbMfwSBaQIERERZx1tXdq1izw3LS1NHTpOKDThvEOHho+fLh++Oen11YXOq1ev
         niTzA33dddedT/ygM3ToUL3yyiu2Y/Gr/gVHrOy4XErvVatWiktLc12DL/Dz1rp8ZiVDY9b6Xz11Vfq
```

Figure 2.7 - Example of a ComplexData output in the WPS's XML output document. In this case the encoding represents an image in PNG format.

2.3 Evaluating the WPS Standard

Since the official publication of the WPS standard in 2007 (version 1.0.0) change requests have been made. These requests are evaluated by the SWG (Standards Working Group) and depending on the outcome of this evaluation and approval of the policy of the OGC's Technical Committee (TC) the requests added to the new version of the standard.

A list of change requests which will be considered for change is published on the OGC website. Also a schedule indicating the development path of the second version of the WPS standard is published. However, up to today no new version has been published, although according to schedule a new version should have been published in 2010.

Besides publications on the OGC website, papers which evaluate the WPS standard have been published in journals. A limited number of these papers actually evaluate the standard as such, most focus on the possibilities of the WPS standard for their own projects.

Michael and Ames (2007) did an extensive evaluation of the WPS standard for use in a client-side desktop GIS. As found in the topic of the research this evaluation focuses on the use of WPS for experienced GIS users (i.e. users who know how to use desktop GIS software packages such as ArcGIS and Q-GIS). In their paper they explain different uses in which application of WPS is convenient, these uses are briefly described below.

2.3.1 Services Under Development

When a tool or service is new and still under development it could be useful to offer a preliminary version as WPS. In this case users can already make use of beta versions of the service and evaluate these by for example reporting bugs to the developers. The advantage is that when the developers publish a new version of the service that nothing has to be done by the user community. The next time the client uses the service, they automatically have the latest version available without the need to update the local system (Michael and Ames, 2007).

2.3.2 As Data Source

The second advice Michael and Ames (2007) give is data dissemination. Since the processing server can also store its own datasets, a WPS can very well be used as a service for retrieving a part of a (processed) dataset. The used dataset can also be dynamic; this is for example the case in when using actual weather data.

Using a WPS as data source makes WPS become more related to other Web Services which are all used for dissemination and updating of data. Depending on the capabilities of the server or a specific process, it could even be possible to store the output data (in case of a geo-referenced dataset or map) directly on a coupled WxS server. Although it is advised to limit the access to such a service to authorized users since this may result in an overwhelming number of new published datasets which may not all be relevant for the service provider.

2.3.3 Need of Processing Power

A final reason for recommending using a WPS according to Michael and Ames (2007) is when there is a great need for processing power. Server configurations often offer more computational power compared to desktop computers, which can reduce processing time when using a WPS instead of the same function on a desktop computer.

However, there is a turning point for using or not using a WPS. Especially in situation where bandwidth of the internet connection is limited, time used for uploading the used dataset can already exceed the total (local) processing time (see also Figure 2.8). Of course this issue does not occur when data from a web service is used.

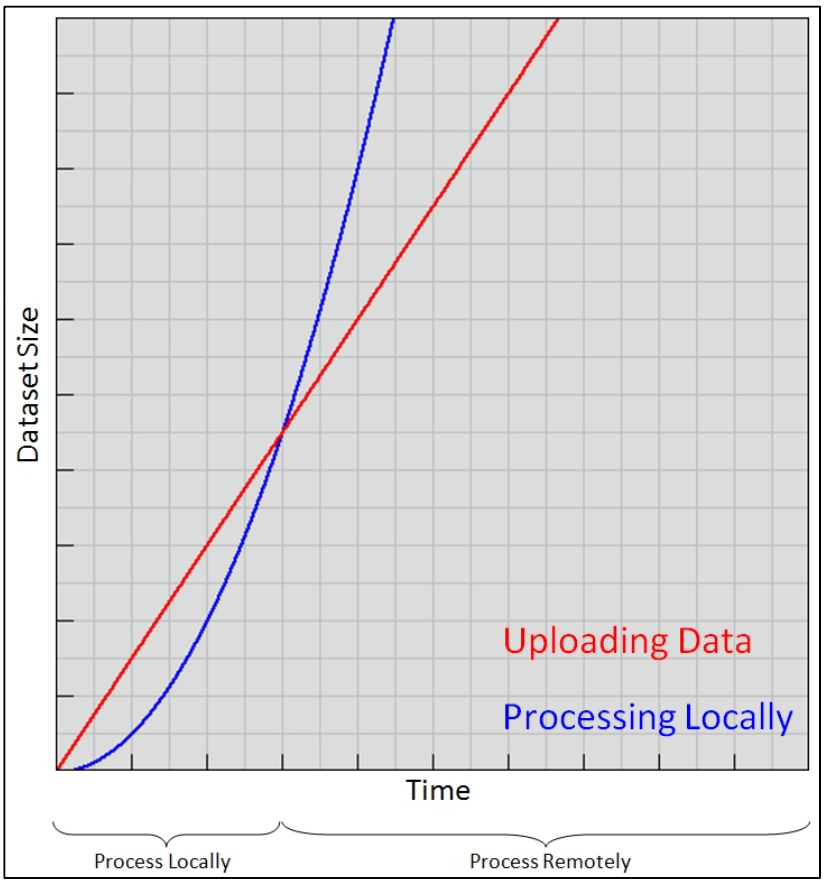


Figure 2.8 - Use local or remote processing according to the uploading time of the dataset. (based on Michael and Ames, 2007)

Other initiatives for WPS like data services come from NASA (National Aeronautics and Space Administration) (NASA, 2012) and ESA (European Space Agency) (Farres, 2009). Both organizations are working on a system for dissemination of data. This is done by means of a processing service which selects the parts of the data requested by the client. Both systems also allow the use of processing services. The NASA system is called NEX and the ESA system is called G-POD. Since there is so much data within the institutions it is not efficient to transfer datasets (data shipping) to clients who like to do analysis with these. Instead of data shipping there is also the possibility of process shipping. In that case the process or model is send to the data provider who runs the script on the dataset (Nativi et al., 2013). Depending on the size of the dataset this may be more efficient compared to data shipping, however, results still have to be returned to the client.

2.3.4 Suggested Enhancements

Besides recommending when to use a WPS, Michael and Ames (2007) also give some recommendations for enhancement of the WPS standard. Some of the most relevant suggestions are mentioned below.

One of the suggestions is to provide a mechanism in the DescribeProcess output which indicates which datasets are available on the server. Especially in cases such as this project it could be interesting for several users to select a different dataset if that would be available. One could think for example of a dataset which is optimized for a certain area of the world or one which requires less processing time.

Another suggestion for enhancement is to improve the number of exception types for errors. A more extensive and complex exception structure would, according to Michael and Ames (2007), improve the possibility for a generic method for handling errors.

2.4 Process Coupling

Coupling multiple WPS processes can be relatively simple. For example in ArcGIS services can be imported as a toolbox and be used as tools. The WPS tools can be used as any other tool, so they can also be imported in the Model Builder where the processes can be coupled to other local and remote tools.

Coupling processes becomes a bit harder if one of the services does not correspond to the WPS standard or when the process is executed on different computing back ends. Findings of these kinds of researches can be very interesting for this research. Besides executing processes from a modelling environment, non-WPS processes on different servers are tried to be executed by following the WPS standard.

Zhao et al. (2012) tries to couple asynchronous workflows in his research. The workflows he researches are asynchronous which indicates that the workflows are not part of a continuous process. In Figure 2.9 an example is given, the client makes a request on which is responded by the service when the client stopped listening.

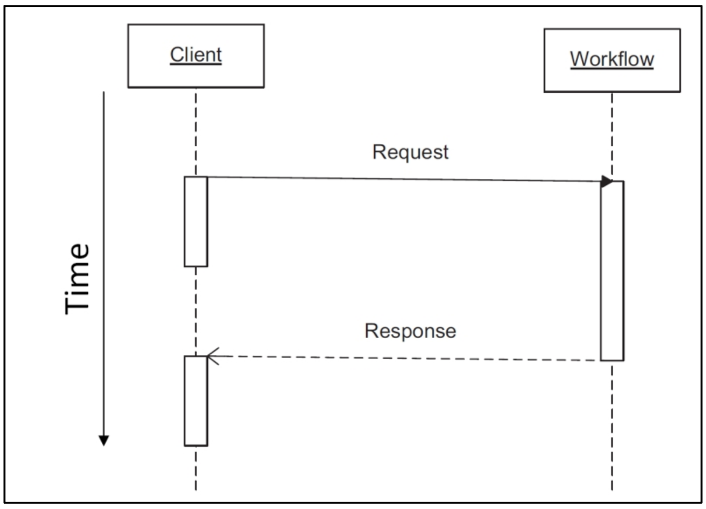


Figure 2.9 - Example of an asynchronous request/response operation. After giving the request the client stops 'listening', when the process finishes a response is sent. (Zhao et al., 2012)

In his paper Zhao et al. (2012) gives different solutions for asynchronous communication. In this project a similar problem is present since a Matlab service has to be started by a WPS, on its turn, the Matlab tool responds by storing an output file in a certain location (see chapter 5) which has to be found by the WPS. In the paper this solution is described as the 'Publish/Subscribe' solution with the use of a notification function (see also Figure 2.10). These notifications, when directly linked to the client can be published for example by sending an email to the client, websockets, or by use of other kinds of online communication.

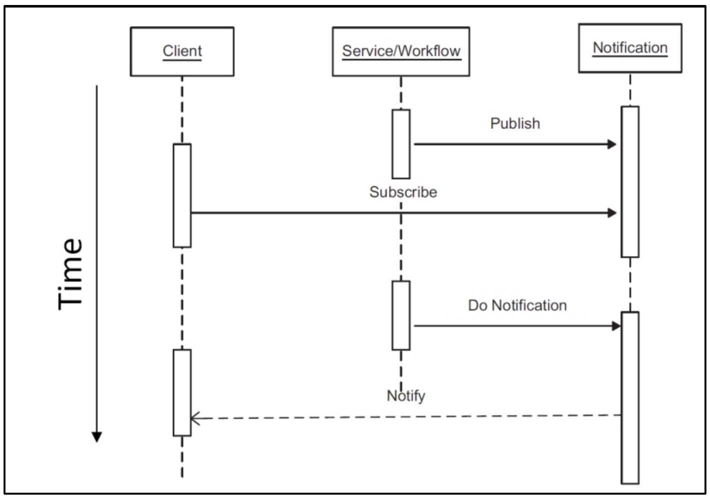


Figure 2.10 - An example of the Publish/Subscribe solution with the use of notifications. The client subscribes to the 'notification' on which the service publishes when the process is complete. (Zhao et al., 2012)

2.5 Projects Using WPS

Since the introduction of the WPS standard several services have been set up for dissemination for example to use on open datasets via viewers or portals, as well for dissemination of different kinds of models. Besides such projects, also ideas for commercial activities of WPS services have been made. In the following sections a number of projects are discussed.

2.5.1 Models as Web Services

Castranova et al. (2013) used the modelling environment OpenMI to predict watershed runoff in an area in western North Carolina (USA). Within the developed model (TOPMODEL) a total of four models are run. One of the models uses a time dependant input which is located on a different server. For retrieving this external processed data a WPS service was used within the modelling environment. Since there was no prepared function for WPS services within OpenMI, a WPS wrapper had to be made. This wrapper could be controlled by OpenMI using the input parameters derived from the previous modelling phase. The wrapper translated these parameters into a request for the WPS which on its turn could be run. The next issue was found in the WPS output which is in the XML, which required some additional descriptors for full integration with OpenMI.

Castranova et al. (2013) concludes that using a WPS in the modelling process does offer some advantages, however it also takes some additional effort to completely integrate the WPS one, the system can be re-used in following projects. Advantages are that the model disseminated via WPS is independent from the overall system so there is no need adapt the computer languages or different parts of the overall (OpenMI) model and there is no need for compiling the model for different operating systems or hardware configurations since the model is deployed at a dedicated server.

A different project using WPS for dissemination of a model is by Dubois et al. (2013). In this project, eHabitat, a multi-purpose WPS for ecological modelling was built for use from a web client. The used WPS model is capable of finding ecosystems with equal properties than which can be specified by the client and is used for monitoring these ecosystems. The eHabitat model was built from Python scripts run from the PyWPS server instance. Besides the WPS service also other services were integrated (i.e. coverages, catalogues, databases and R integration (statistics)). Using WPS for eHabitat offers some benefits for example that different data types can be relatively easy used in the model and for different types of end-users the model does not have to be changed, only the appearance of the web client.

An interesting initiative for disseminating models as Web Services is by Nativi et al. (2013). In his paper he describes how models can be set up in the cloud and coupled to improve access to available models and interoperability between models and modellers. One of the technologies considered to be valuable for increase of interoperability is WPS. Still there is a long way ahead of a final ‘Model-Web’ but first promising and beneficial steps have been taken.

2.5.2 Geostatistics via WPS

In risk management for example, geostatistics are commonly used nowadays. Special tools and software has been developed, for example ‘Automap’. Automap is an open source algorithm written in the programming language R (often used for statistical analysis). Jesus et al. (2008) worked on a project to create a web based version of automap using WPS.

In the project Jesus et al. (2008) chose to use PyWPS for disseminating this R based algorithm. Such operations give an interesting insight in how to execute a non-Python algorithm from a Python script. In the project the Python module RPy was used. RPy allows the users to access R functions and sessions from Python.

On a technical point of view, Jesus et al. (2008) conclude that the use of XML can be a bottleneck in the efficiency of the system. Although XML is used as a standard for output in WPS, it can become rather inefficient to send an XML file with many points as output since these files consume a significant amount of system resources.

2.5.3 Commercial Interest

The use of WPS or other web services can reduce the amount of required computational power in companies and institutes dramatically when cloud computing services are used. Baranski et al. (2010) tries to find a solution for a revenue model for service providers. The main question of the paper is to find a way to extend the WPS specification or to integrate a licensing system for charging consumers for using the services.

Two different payment models are identified by Baranski et al. (2010). A flat rate, which indicates that the consumer pays a fixed amount for using the service or a pay per use option where consumer only pays for the time a service is used. In the paper a licence model is introduced where consumers can reserve time slots on a server using the 52North WPS instance. The payment model would not be limited to use for geoprocessing services but a wide range of 'cloud services' can be coupled to a certain licence system.

2.6 Other Methods for Remote Processing

Besides using WPS there are many methods for running remote processes. One of the best known technologies that allows dynamic data requests and execution of (specific) programs is the 'Common Gateway Interface' (CGI). Another (former popular) technology is the 'Remote Procedure Call' (RPC), usage of this technology declined. CGI can operate with many scripting languages such as Python, PHP ('PHP: Hypertext Pre-processor') or Bash.

A method apart from CGI is SOAP (Simple Object Access Protocol). SOAP is a lightweight protocol for data exchange between clients and servers. For communication SOAP uses XML documents which can be send using HTTP and other protocols like FTP (File Transfer Protocol) and SMTP (Simple Mail Transfer Protocol) (W3C, 2007). SOAP can also be used to send WPS requests to servers. This allows to use protocols other then HTTP, which can be an advantage for long running processes where communication can be done via email.

In computer science different layers of communication are identified in the Open Systems Interconnect (OSI) standard. Communication via HTTP is in the top level of the OSI 'layer system', the application layer. CGI and SOAP are standard which are operated using HTTP so these are actually inside or on top of the application layer. A schematized overview of OSI is in Figure 2.11.

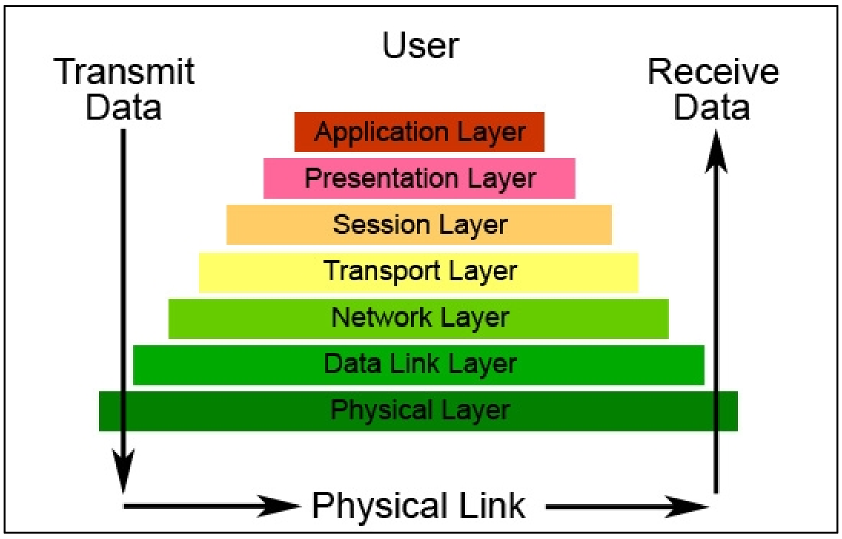


Figure 2.11 - Schematized overview of the OSI model. HTTP is located in the Application Layer, CGI and SOAP are inside or on top of this. (Source: University of Washington, www.washinton.edu)

The main difference with WPS and other methods for remote processing is that WPS prescribes the way requests are made and how the server should respond. In other methods the structure of requests (when using URL's for that purpose) is structured in an equal way as WPS requests (see also 2.2.1), however, the key words are completely free to be defined by the programmer.

3. WPS System Setup

This chapter describes some of the essential parts required for working with WPS. The base for setting up a service is the choice for a certain software package. A short comparison and consideration is made in 3.3.

3.1 Server

To finally get the WPS processes running, a server system has to be setup. The setup process contains different phases such as finding and testing the server software and setting up a web server connected to the internet. To get this done a server implementation has to be chosen first. This is a software package that actually makes it possible to run processes using the WPS standard. Different server instances are tested on a basic Windows PC to find out the working of the system. In this phase also example scripts for processes are tested and studied for their build up.

With experiences from the testing phase and information found from previous studies, a server instance has to be chosen. Criteria to this instance are at least the possibility to use the Python language for programming services and stable operability during the testing phase.

The final step that has to be made after choosing the software, is installing the web server. The used server will be a Deltares server especially setup for testing different kinds of online software and services. Although the server is located at Deltares, for safety reasons it is not connected to the local network so operations has to be performed via the internet. The server is running on the Debian operating system (Unix like system) which can be accessed from the Deltares intranet using PuTTY, a terminal. The web server runs NGiNX and Gunicorn server implementations.

3.2 WPS Clients

Working with requests from the address bar of a web browser is not the most user-friendly method for accessing information. For that reason a number of client interfaces have been developed for integration in desktop GIS software but also as API (Application Programming Interface) on websites.

For simplified operations with URL requests, a so called ‘Request Builder’ can be used. A Request Builder is a service, usually offered by the provider of the WPS via a website. The Request Builder can for example be a tool to build URL’s for server requests or, a bit more advanced, build complete XML files for use with HTTP Post. An example is offered as demonstration by 52North[2].

A user friendly interface that can be used is an API. Using an API for dissemination of a WPS offers the user a complete package of base maps, project specific maps and (WPS based) analysis tools. An advantage is that users do not have to install specific software on their computer to make use of the service.

An example of an online WPS client is from the eHabitat project (Dubois et al., 2013). In this project a client was built capable of combining different inputs and generating new layers using available data via the WPS standard. In this project the developers made use of an available software package (PyWPS) for server implementation. The inputs are processed using Python and R scripts, results are published via WMS in the same online client.

More advanced users, such as GIS experts who are used to working with desktop GIS's can use (most of) their favourite software packages. In case of using a desktop GIS a WPS is for example added as a toolbox. In this case the user can use (selections of) existing layers and parameters for input of the process. The WPS output is directly added as new map layer and can be used for further analysis. Coupling of WPS processes and existing tools can be done relatively simple in this way. In ArcGIS for example, WPS processes can be used in the Model Builder. To test processes built for this project the QGIS WPS plug-in is used. QGIS is an open source desktop GIS and is freely available. QGIS was chosen because of the ease of distribution (no licenses like ArcGIS). QGIS however offers a less advanced feature compared to the ArcGIS WPS implementation. The QGIS plug-in only allows the execution of a single process, no visual process builder is available which allows for process chaining.

3.3 WPS Server Implementations

Different organizations have been developing WPS server implementations from which most are freely available on the internet. The implementations mainly differ in the way they are installed on the web server and the programming language supported for processes. The choice for a certain implementation was not based on based on an extensive number of requirements. The only requirement from the view of the project is that the implementation should be able to disseminate Python processes as WPS. A requirement from a more practical point of view is that the implementation should be relatively easy to be installed and work stable.

For this project two implementations were tested extensively, 52North and PyWPS. Both implementations support Python processes according to the documentation. Both implementations and the experiences with them are described below. Other implementations (which are not tested extensively) are for example Geoserver WPS[3], ZOO-

[2] http://geoprocessing.demo.52north.org:8080/wps/test.html
[3] http://docs.geoserver.org/2.3.3/user/extensions/wps/index.html

Project[4] and Deegree[5]. These instances do not meet our specific operational demand since these are Java based instead of Python. Although it is possible to execute Python script under Java, for example with the use of the Jython (Python) library, this takes more effort considering system setup and process implementation than using Python code directly.

3.3.1 52North

52North provides a Java based (Open Source) WPS implementation which can be used with Apache Tomcat. Installation of the package is very easy (mainly copy-paste) and takes only a few minutes. The implementation works smoothly with the Q-GIS WPS plug-in but can also be approached by simply using a web browser. An advantage for users interested in basic GIS functions in a WPS is that the 52North WPS comes with a lot of ready to use processes.

Custom made processes can be added by using the web interface or by copy-pasting and modifying the configuration file. For custom Python processes some additional changes have to be made, some specific folders should be added and certain configuration parameters should be changed (52North, 2013).

Originally Python support in the 52North WPS implementation was designed for use with ArcGIS-Server and ArcPy (ESRI's ArcGIS Python library). However, according to the provided documentation it should be possible to use custom Python processes without making use ArcGIS-Server and ArcPy. Unfortunately in the testing period Python support did not seem to be operating correctly, not even with the example Python processes provided by 52North. The developer community was contacted but unfortunately a solution could not be found. The 52North WPS implementation however looks very promising and the software is regularly being updated which shows that the community is still working on improving the system.

3.3.2 PyWPS

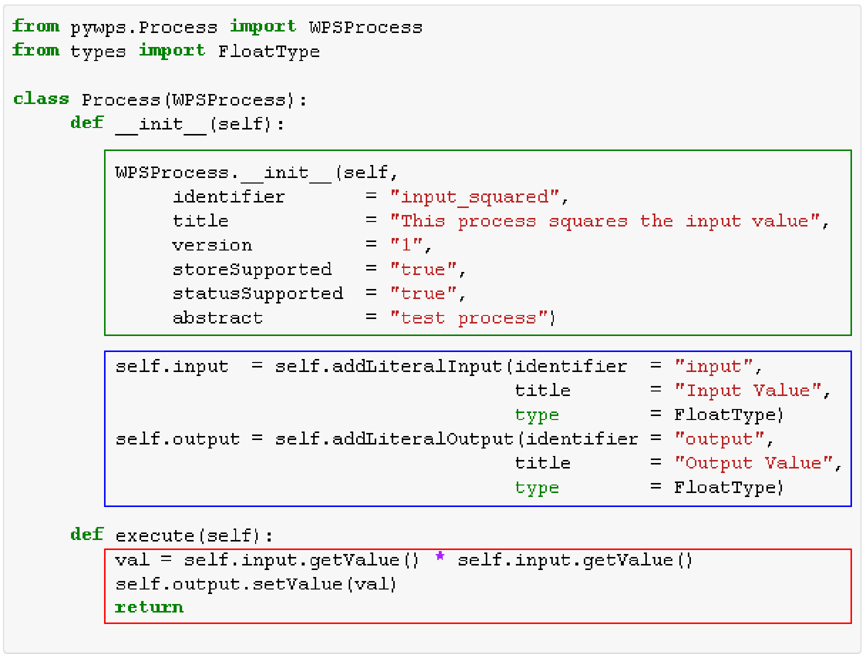
The PyWPS instance is fully written in Python and was originally developed for GRASS GIS (open source GI software) operations. GRASS offers high quality GI functionalities which can be fully controlled from Python and allows for easy integration in PyWPS (Neteler et al., 2012). The first version was launched in 2006 and supported WPS version 0.4.0. With development of the WPS standard PyWPS grew along and currently supports the latest version 1.0.0 (PyWPS, 2011). At the FOSS4G conference in 2011 PyWPS was awarded as the WPS instance that provided best interoperability in a WPS-Shootout (Garnett and Fenoy, 2011).

Installation of PyWPS is relatively simple when CGI (Common Gateway Interface) (standard supported in Apache) is used. The downloaded files can be copied to a location on the server and the CGI script which refers the server to the specific PyWPS folder has to be added to the

[4] http://www.zoo-project.org/docs/kernel/index.html#kernel
[5] http://download.deegree.org/documentation/3.3.1/html/intro.html#characteristics-of-deegree-wps

Apache CGI folder. Originally, PyWPS was developed to use under Linux operating systems. The instance, however, can also be used under Windows. For the sake of this thesis project a tutorial has been written and is included in appendix I.

In PyWPS Python scripts are directly executed as WPS processes, recoding of existing scripts to PyWPS processes is relatively simple. This is a great advantage for people who already wrote their own Python scripts which are afterwards disseminated as WPS process, example script is in Figure 3.1. More details on the structure of PyWPS processes are in Appendix 0.



```
from pywps.Process import WPSProcess
from types import FloatType

class Process(WPSProcess):
     def __init__(self):

         WPSProcess.__init__(self,
              identifier       = "input_squared",
              title            = "This process squares the input value",
              version          = "1",
              storeSupported   = "true",
              statusSupported  = "true",
              abstract         = "test process")

         self.input  = self.addLiteralInput(identifier  = "input",
                                            title       = "Input Value",
                                            type        = FloatType)
         self.output = self.addLiteralOutput(identifier = "output",
                                            title       = "Output Value",
                                            type        = FloatType)

     def execute(self):
         val = self.input.getValue() * self.input.getValue()
         self.output.setValue(val)
         return
```

Figure 3.1 - Example script of a PyWPS process that squares the input value. The green box contains metadata of the service and server based parameters; the blue box contains the definitions of in- and outputs; the red box contains the actual operation.

3.3.3 Choosing a Suitable WPS Server

After a round of exploring different server instances, PyWPS seemed to be the best package for this project. The local installation could be used for testing purposes, for server application the WSGI (Web Service Gateway Interface) version of PyWPS is used. WSGI offers a better performance when it comes to running multiple (Python) processes compared to CGI. An overview of server instances is in Table 3.1.

Name	Protocol	Languages	Stable operation	Remarks
PyWPS	CGI or WSGI	Python	Yes	Python based
52North	CGI	Python, R, Java	No	Promising package
ZOO Project	CGI	Java	Not tested	
Geoserver	CGI	Java	Yes	
Deegree	CGI	Java	Not tested	

Table 3.1 - Overview of WPS server instances and findings.

4. Case 1 - Tide Forecasting Service

This chapter describes the tide forecasting case. Prior to a description of the practical work an introduction is given on measuring tides with different methods, analysis of these measurements and the method of forecasting tides.

4.1 Tides: Measurement and Analysis

The global tide is actually a long-period wave generated by the gravitational forces on the Earth's oceans exerted by the Sun and the Moon. The water levels during the ebb and flood periods (low and high water) are depended on the period in the tidal cycle (spring and neap tides) and the shape, size and depth of the 'basin' (Wright et al., 1999). For example in the French-British Channel, tidal water level is locally amplified due to wave resonance and shoaling (Bosboom and Stive, 2011). This local difference in water level is unique for every location in the world.

Measurement of Tides

Tidal periods have been recorded for long periods of time with special tidal gauges. Most of these measure points are concentrated in areas close to the shore, usually at ports (see also Figure 4.1). Offshore measurement points are limited and usually expensive to setup since special platforms or buoys have to be placed. Still the number of (on- and offshore) measurements points are limited, so conversion tables are often used to derive water levels at other locations nearby (Pinet, 2006). Some gauges (or at least the locations of measurement) are up and running for several centuries (e.g. the water levels of Amsterdam (the Netherlands)) have even been measured from the 18th century). These long periods provide good opportunities for analysis of tidal oscillations.

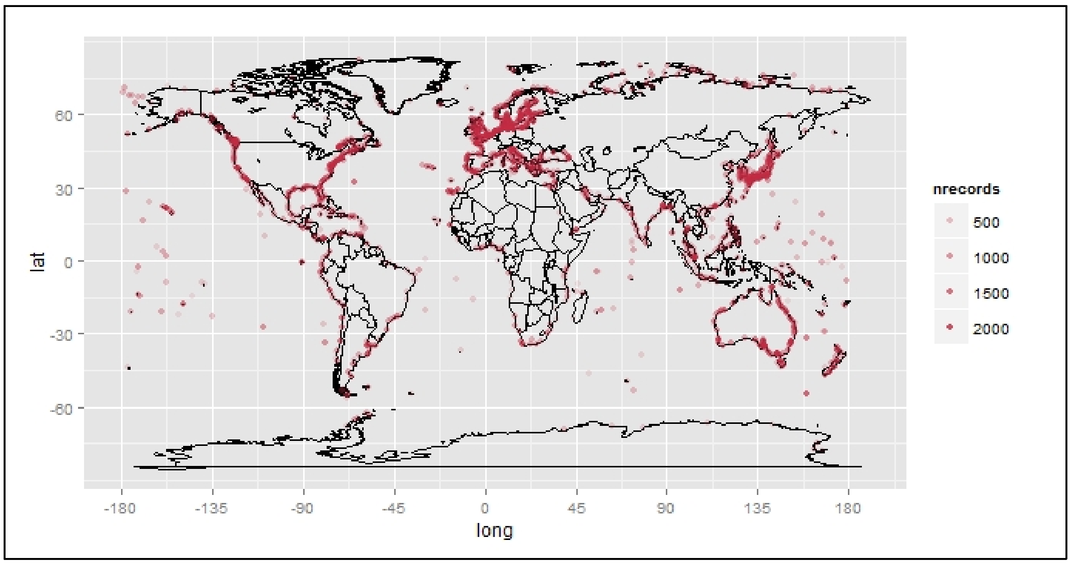


Figure 4.1 - Overview of tidal stations over the world. Map based on dataset of Woodworth and Player (2003).

The easiest method of measuring the tide is to use a (large) ruler mounted to a fixed object such as a pier. Tidal levels can then be recorded by writing down the water level and time. This method however is not practical since usually tidal recording is often needed in long term periods. For long term recordings tide gauges are used which can record tide levels automatically, and these days, in digital format for easy dissemination. The tide gauge is a semi-closed well (to limit wave influence) with a floater connected to a recorder. A simplified image of the working of a tide gauge is shown in Figure 4.2. (Pinet, 2006)

The accuracy of tide gauges is usually quite good; with hourly measurements an accuracy of 1 to 2cm can be reached. Improvement of the tide gauges can be obtained when the mechanical parts such as the floater are replaced by remote sensors which measure the water level in the well. The accuracy of measurements can be improved to 1 to 2mm. (Steenbergen, 2004)

Modern day tide gauges, in comparison to older ones, can record the tidal level continuously and usually publish the measured water level directly online. In the Netherlands for example data measured by offshore platforms can be accessed online (e.g. live.getij.nl). These data however still contains only point measurements at fixed locations. Measuring tides further offshore or outside the measuring platforms or buoys should be done differently.

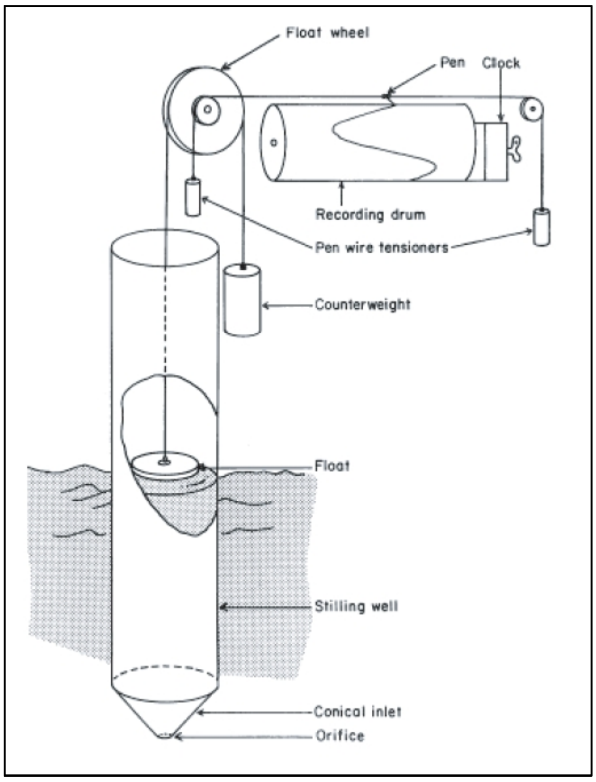


Figure 4.2 - Schematic image of the working of a (classic) tide gauge with analogue recording. (Source: National Oceanography Centre UK, noc.ac.uk)

4.2 Remote Sensing of Water Levels

Tide gauges do not cover for large areas in the oceans. For that reason, satellite altimetry is used. With the use of satellite measurements it is possible to measure water levels over a large area of the Earth. Water levels, measured with radar techniques, are relative to a geographical reference system (e.g. WGS84); tide gauge measurements usually measure water levels relative to national datums (e.g. the Dutch NAP (Amsterdam Ordnance Datum).

One of the oceanographic satellite missions was the TOPEX/Poseidon (T/P) mission (other missions are JASON 1 and 2). This satellite was launched in 1992 and had the primary goal to increase knowledge about ocean circulation and its influence on the Earth's atmosphere. The T/P mission ended after almost 13 years and almost 62,000 orbits due to a technical malfunction (NASA, 2006). The mission however was only planned to last for five years but finally became the longest Earth-orbiting radar mission (NASA, 2006).

To improve accuracy of the measurements, T/P used several tracking systems (i.e. Satellite Laser Ranging (SLR), the Global Positioning System (GPS), Doppler Orbitography and Radio positioning Integrated on Satellite (DORIS)). Due to these systems T/P could reach an accuracy of up to 2.5cm, which was a great improvement compared to its ancestors, for example Geosat which had an accuracy of 10-20cm which was not very suitable for oceanographic investigations. (Steenbergen, 2004)

The T/P satellite was able to cover about 95% of the Earth's ice-free oceans, up to a latitude of (plus and minus) 66 degrees. The repeating period, the period in time to cover the area possible to cover, was just under 10 days. An example of the (processed) data output of T/P can be found in Figure 4.3; the image shows the sea level deviation during an El Niño event.

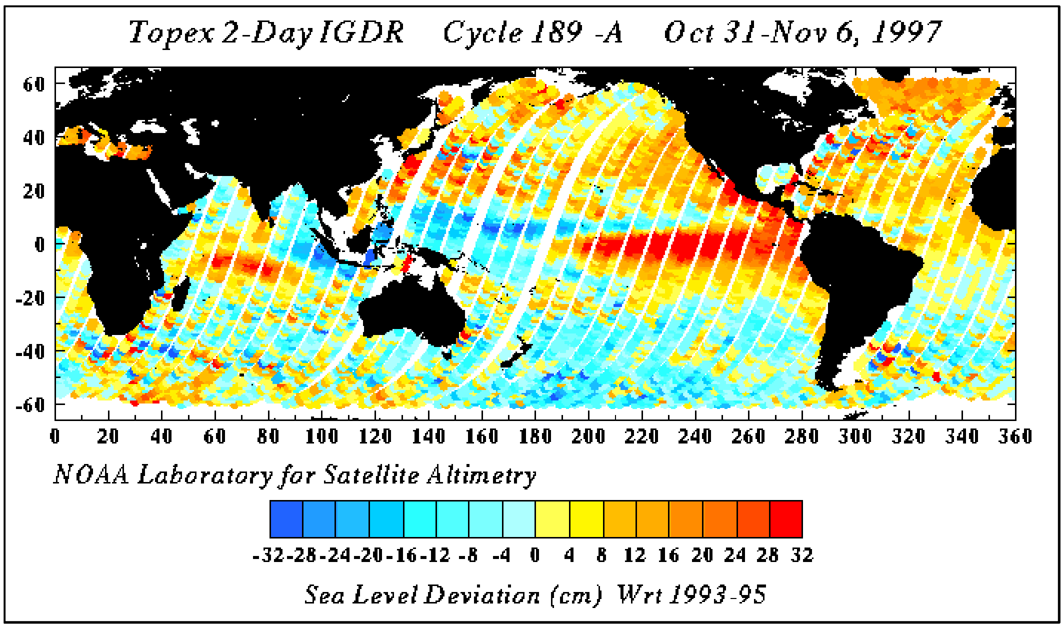


Figure 4.3 - Example of the output of the T/P imagery. Image shows the water levels during the El Niño event in 1997. (Source: Penn State University, www.e-education.psu.edu)

4.3 Forecasting the Tidal Wave

For predicting the tidal curve, one should know which components actually cause the variations in water level. The tidal wave is a combination of many harmonic constituents, in total there are 390 components identified. All these different components have different periods and ratios of influence on the mixed tide (total tidal wave) (Wright et al., 1999).

Tidal components follow the shape of a harmonic equation like in Equation 4.1. Variables *A*, *w*, *t* and φ represent the amplitude (*A*) [meters], speed (*w*) [degrees per hour], time (*t*) [hours] and phase (φ) [degrees] (difference in harmonic phase compared to the current phase in the tidal cycle) of the different components. (Phillips, n.d.).

$$f(t) = \sum A \times cos(w \times t + \varphi)$$

Equation 4.1 - Water level (f) generated by astronomical influences. General equation for calculating the tidal curve from all constituents.

The different tidal components are added together and the complete tidal curve is calculated. Addition of components is done in different ratios since all components have a different influence on the mixed tide. The highest influence comes from the forces generated by the Moon (Principal lunar, M2 tide); a second place is for the forces generated by the Sun (Luni-solar, K1 tide) (Wright et al., 1999).

An example of how such a mixed tidal curve may look like is shown in Figure 4.4. The graph shows the M2 and K1 tide together with the corresponding mixed tidal curve which is composed out of 11 tidal components.

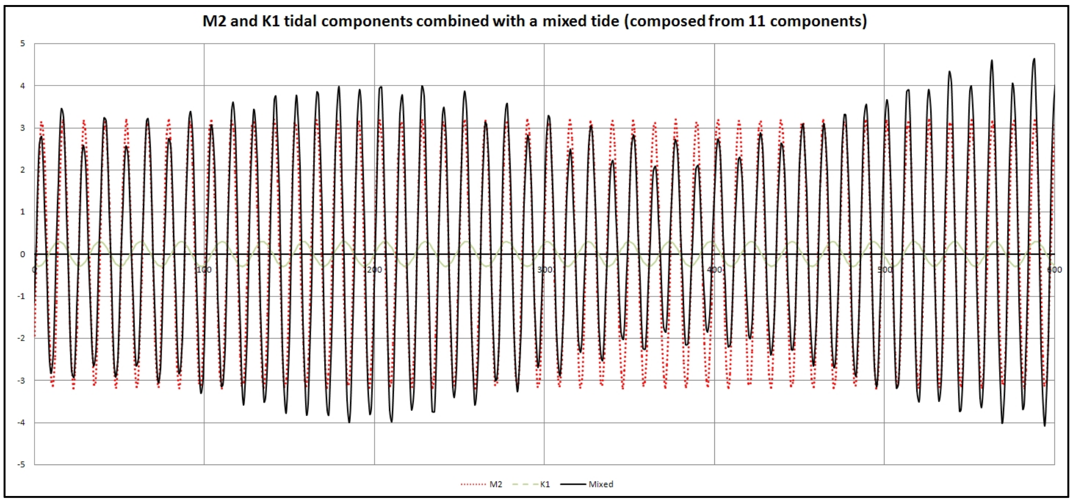


Figure 4.4 - M2 (red line) and K1 (green line) tidal constituents combined with a mixed tide (black line) composed from 11 constituents, other constituents are not shown for clarity reasons. X-axis shows time [hours], the Y-axis shows the water level [m] at a fictional location.

A dataset with parameters is only suitable for one specific location since tidal influence is unique for every location (e.g. due to differences in phase, depth and basin shape), usually where a tide gauge is located. The available measured time series from the gauge are used for extraction of the different constituents. This extraction is usually performed with an analytical software package which uses a harmonic (sinus fit) or Fourier analysis. It is advised to use a time series of a long period, say 20 years, for the analysis; this will result in more reliable results because then also nodal factors can be analysed. Nodal factors correct for differences in the phase cycle (18.6 years). A software package that could be used is for example T_TIDE* (Pawlowciz et al, 2002). In this project nodal factors are not taken into account because this requires quite some more programming operations. For operational future applications these factors should be included.

With the introduction of time series from satellite measurements the datasets are no longer bound to a single location, although the grid size is relatively coarse. The dataset (TPXO7.2) used in this project offers a 0.25 degree (in longitude and latitude) resolution and a global coverage. Use of this dataset allows the user to make predictions for the future tides globally.

4.3.1 TPXO7.2 Dataset

For the tide forecasting services a dataset is used as an input file for all requests. The dataset used originates from the Oregon State University and is derived from the TOPEX/Poseidon (T/P) dataset (please see section 4.1 for more information on T/P and tidal measurements and analysis) (Egbert & Erofeeva, 2002). The version of the 'TPXO' dataset which is used is version 7.2 which was published in 2009.

The dataset contains a total of 13 tidal constituents with corresponding amplitude and phases derived from the T/P dataset. This number is limited (in total there are 390 constituents) due to the limited survey period of T/P, which is almost 13 years. However, since quite some important (high ratio) constituents are available in the dataset it is expected that the quality of a tidal prediction based on this dataset will be sufficient for the sake of this research, demonstrating the set up a tide forecasting WPS. For projects where higher accuracies are required, higher resolution datasets can be used. The accuracy required for this project is returning a correct shape of the tidal curve (like in Figure 4.4), accuracy of water levels is in this case less important. An example of the global amplitude maps is shown in Figure 4.5.

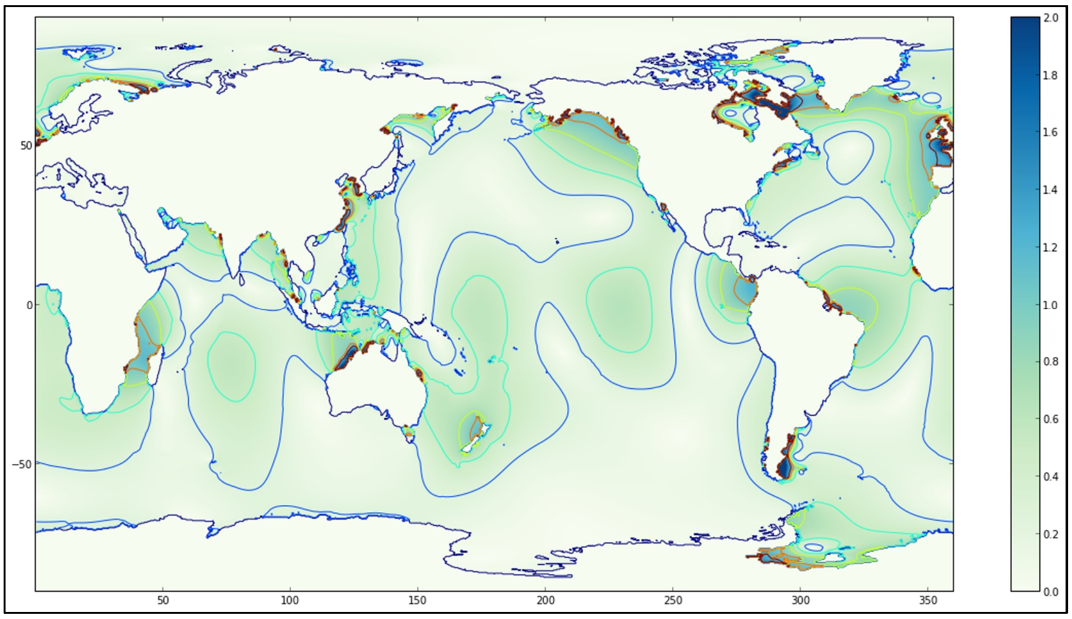


Figure 4.5 - Amplitude map [in meters] for the M2 (Moon) constituent plotted from the TPXO 7.2 dataset.

For derivation a specially developed software package, OTIS (Oregon State University Tidal Inversion Software), was used. This package uses a least squares analysis to derive all components. Other tidal prediction software, for example T_TIDE* which is often used, uses the same method for analysis (Pawlowciz et al., 2002).

The data is published in netCDF (OGC, 2011) format, an open (OGC standard) format which is gaining popularity among metocean projects and is used as the standard for data at Deltares. The resolution of the dataset is 0.25 degree.

For calculations with the dataset one needs a reference time for the phases since these are time dependent parameters. The reference time of the TPXO dataset was set at January 1 (at 00:00:00hrs GMT) in the year zero of the Julian calendar. The Julian calendar was introduced by Julius Caesar in 46 B.C. However and was in use until the year 1582. From then the Gregorian calendar has been used. The difference between these calendars is about thirteen days which have to be taken into account when converting dates from one to another.

Recently a new version of the TPXO dataset was released; version 8. This ‘atlas’ is also online available and offers increased resolution (1/6 degree in open oceans and 1/30 degree in validated areas) and better performance in coastal regions since this dataset is validated to tidal stations. However, there is a disadvantage to this dataset, the total size of this dataset, which is split in nine separate files for each constituent is four gigabytes. The size of this dataset is expecting to limit the performance of the service and for the purpose of this project the increased resolution is not required. However, for projects involving much larger datasets (in the sense of terabytes) other solutions should be found.

The dataset is re-served from the Deltares OPeNDAP (Gallagher et al., 2004) server[6]. Doing so preserves the working of the service when links become broken. Also this reduces the amount of data stored locally at the processing server which is a second option for securing the location of a dataset.

4.4 Tide Forecasting Service

The first service being built is a tide forecasting service. The goal of this service is to disseminate a global scale tool which can predict water levels due to astronomical influences. The tool is built in different steps since this is the first to be programmed for use via WPS. A first step is a simple look-up tool for tidal constituents only, after this step a forecasting tool is made. In a final step some more advanced functionality is added to the service (section 4.4.3).

4.4.1 Constituent Look-Up tool

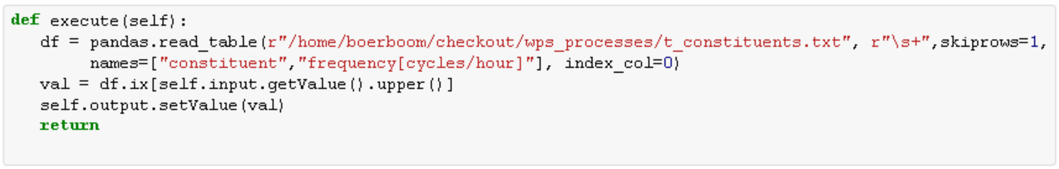
The first WPS process that is made is a look-up tool for tidal constituents. In this tool the user gives the name of a constituent as an input, the result is the period of the constituent in cycles per hour found in a separate file.

The actual script is very simple and exists of only three lines of code. However, this offered a good exercise for working with the WPS script structures. It showed that creating an actual process is relatively easy and the service itself can offer fast responses to requests.

[6] http://opendap.deltares.nl/thredds/catalog/opendap/osu/tpxo/catalog.html

This process was tested locally as a practical exercise for setting up a simple process. Later on this process was also disseminated from the test server where it was adapted to the Debian (Linux) environment.

The service works as following. In the input request the name of a constituent should be given. With this name the script searches in an external text file containing a table of constituents with corresponding values. As an output, the service returns a message to the client with the name of the constituent and the corresponding value.



```
def execute(self):
    df = pandas.read_table(r"/home/boerboom/checkout/wps_processes/t_constituents.txt", r"\s+",skiprows=1,
           names=["constituent","frequency[cycles/hour]"], index_col=0)
    val = df.ix[self.input.getValue().upper()]
    self.output.setValue(val)
    return
```

Figure 4.6 - Actual code used to look up the value of a tidal constituent in a table; ‘df’ defines the actual table (dataset) and ‘val’ is the result of the lookup function.

4.4.2 Basic Forecasting Tool

The basic forecasting tool provides a single water level at single location at a specific moment in time. This tool is used as a starting point for following services and so contains all required steps for making a water level prediction based on the TPXO dataset.

The working of the tool is basically relatively simple since the available dataset is quite extensive. In the dataset the amplitudes and phases for each available constituent are given. With these values the total tidal curve and the time dependant water level according to the astronomical tide, can be rebuild. A schematic view of the forecasting tool is in Figure 4.7.

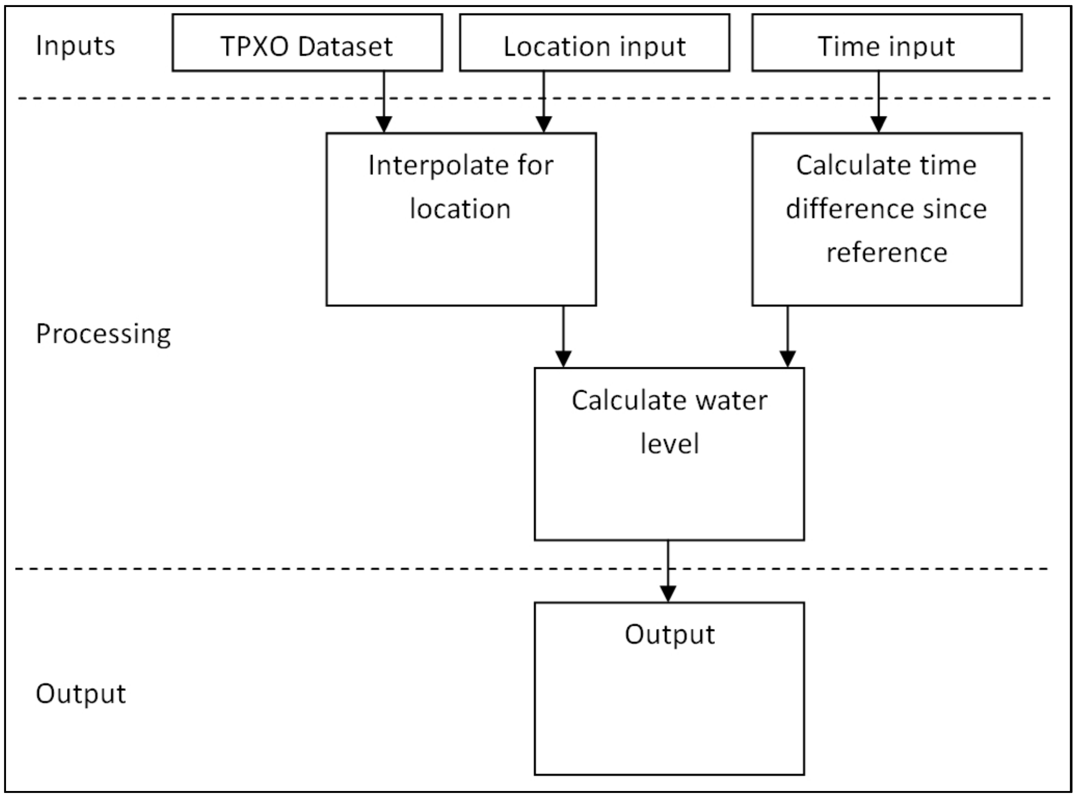


Figure 4.7 - Schematic view of the Tide Forecasting service.

There are some issues that should be taken into account when constructing the tidal curve. One should for example consider date and time calculated from the starting point of the dataset, discrete values in the dataset versus continuous values in requests and the units of all parameters involved.

A second issue that should be considered is that the dataset only contains a fixed point grid where values are known. When a location in between these values is given as input the process will be unable to execute the process since no data for that location is available. For that reason an interpolation will be made using the nearest surrounding grid points (4) when a location does not fit into the grid.

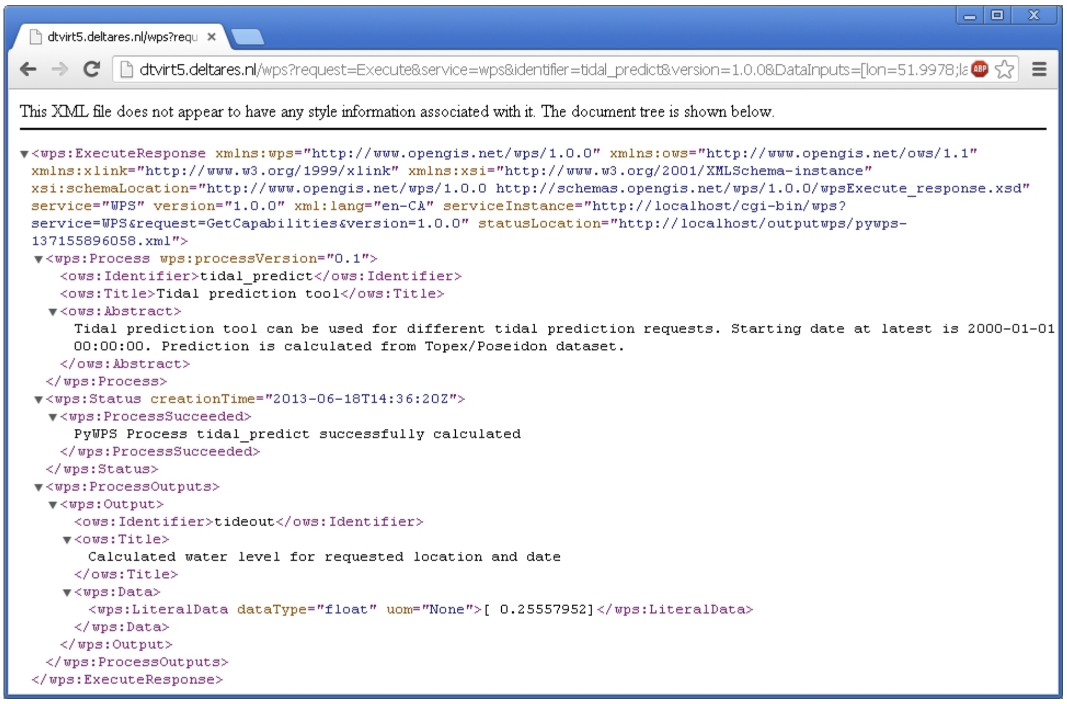
For the interpolation operation the bivariate spline approximation for a rectangular mesh is used (SciPy library Python function). For this operation the amplitudes (Z direction) are interpolated but also the phases (X and Y directions) are interpolated. To get these last ones, the phase variable should be split into an X and Y component which can then be interpolated separately.

A last issue to be considered are the units of all used variables and parameters. Time for example is in GMT (Greenwich Mean Time); depending on the time of the year the difference between the Dutch time and GMT is one or two hours, depending on the application of daylight savings. The used reference system is WGS84 and the water level is given in meters.

For testing purposes a web browser is used, in Figure 4.8 below the Execute URL is given, the different parameters can be adjusted to the client's preferences. In this case a browser has an advantage over the QGIS plug-in since the browser returns the complete XML file (see Figure 4.9) where the plug-in only shows a stripped version.

```
http://dtvirt5.deltares.nl/wps?
     request=Execute&
     service=wps&
     identifier=tidal_predict&
     version=1.0.0&
     DataInputs=[
           lon=51.9978;
           lat=3.2750;
           year=2013;
           month=8;
           day=29;
           hour=12;
           minute=0
           ]
```

Figure 4.8 - Execute URL for the tide forecasting service, 'tidal_predict'. Under DataInputs all input parameters are specified, in this case the location in longitude (lon) and latitude (lat) and the moment in time specified in the parameters 'year', 'month', 'day', 'hour' and 'minute'. This execute request calculates the water level on August 29 at 12 o'clock [GMT] at 51.9978N, 3.2750E.


dtvirt5.deltares.nl/wps?requ
dtvirt5.deltares.nl/wps?request=Execute&service=wps&identifier=tidal_predict&version=1.0.0&DataInputs=[lon=51.9978;la

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd"
 service="WPS" version="1.0.0" xml:lang="en-CA" serviceInstance="http://localhost/cgi-bin/wps?
 service=WPS&request=GetCapabilities&version=1.0.0" statusLocation="http://localhost/outputwps/pywps-
 137155896058.xml">
  ▼<wps:Process wps:processVersion="0.1">
     <ows:Identifier>tidal_predict</ows:Identifier>
     <ows:Title>Tidal prediction tool</ows:Title>
    ▼<ows:Abstract>
       Tidal prediction tool can be used for different tidal prediction requests. Starting date at latest is 2000-01-01
       00:00:00. Prediction is calculated from Topex/Poseidon dataset.
     </ows:Abstract>
   </wps:Process>
  ▼<wps:Status creationTime="2013-06-18T14:36:20Z">
    ▼<wps:ProcessSucceeded>
       PyWPS Process tidal_predict successfully calculated
     </wps:ProcessSucceeded>
   </wps:Status>
  ▼<wps:ProcessOutputs>
    ▼<wps:Output>
       <ows:Identifier>tideout</ows:Identifier>
      ▼<ows:Title>
         Calculated water level for requested location and date
       </ows:Title>
      ▼<wps:Data>
         <wps:LiteralData dataType="float" uom="None">[ 0.25557952]</wps:LiteralData>
       </wps:Data>
     </wps:Output>
   </wps:ProcessOutputs>
 </wps:ExecuteResponse>
```

Figure 4.9 - Output XML file of the Tide forecast tool. The output value is in between the 'wps:LiteralData' tags.

4.4.3 More Advanced Functionalities

The basic service allows calculation of the water level for a single location at a specific moment in time. Since the service is not yet optimized for processing speed, requesting a single water level is probably too time consuming (15 seconds) for most users, who are likely to prefer the water level over a longer period as outcome. An optimization can be made in the implemented script and location of the dataset. The relatively long running time is independent of the choice for WPS. However, depending on the type of user and the proposed application of the data different tools can be created. An overview of implemented processes and their properties is in Table 4.1.

Process	Function	Output
constituents_lookup	Finds the value of a tidal constituent.	String with name van corresponding value of the constituent
tidal_predict	Forecasts the water level at a single time and location	Single value
tidal_predict_range	Forecasts the water level at a single location for a specific time range	Range of water levels and time stamp as array type
tidal_predict_graph	Forecasts the water level at a single location for a specific time range	Graph indicating water levels over a period of time
tidal_predict_txt	Forecasts the water level at a single location for a specific time range	Plain text containing water levels and time stamps
t_p_graph_circ	Forecasts the water level at a mobile location for a specific time range	Graph indicating water levels over a period of time and track

Table 4.1 - Overview of functionalities of implemented tide forecasting services.

Knowing the water level at a single moment in time for a single location is, expected for most users, too limited. One lever above the single calculating a single water level is a range of water levels for a specific location. This can for example be interesting for performing offshore construction works which are limited for current speeds but also for people practicing aquatic sports such as surfers or scuba divers. For those groups the service was adapted such that a range of water levels for a given period of time is returned as a graph, see Figure 4.10. Water levels shown in the graph are still related to a single location.

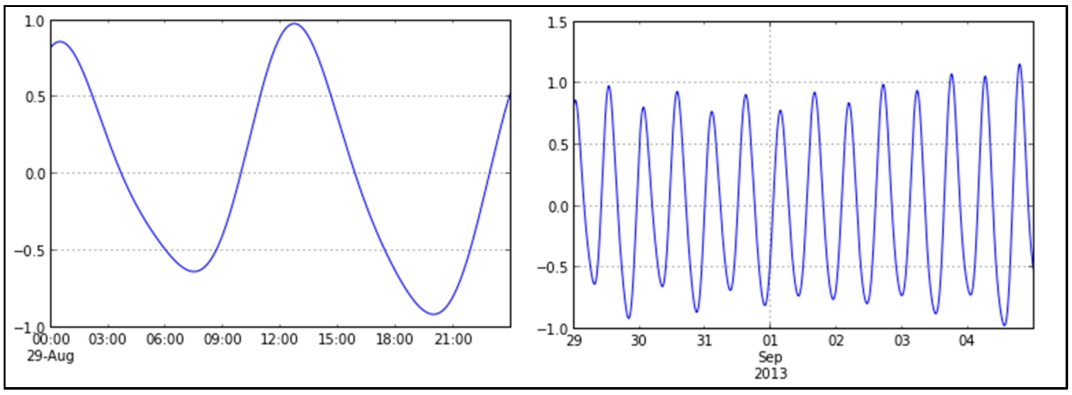


Figure 4.10 - Forecasted lapse of the water level close to the entrance of the Port of Rotterdam on August 29 2013 (left) and from August 29 until September 5 2013 [GMT] (right) (51.9978N, 3.2750E, the location of Europlatform 1).

Besides variation in time, variation in location can also be required by the user. A different service implemented accepts the radius of a circle as input. With this radius, corresponding starting location and period of time the water level expected to be observed during a journey is given. An example of such a graph is given in Figure 4.11. In the figure the track is shown on the right and on the left one finds the forecasted water levels on the way. Notice the period without water level variation, only noise is forecasted, this is the result of calculating the water levels on the British and French mainland. Instead of a track in circles, the script can be adapted as such that it allows vectors as input.

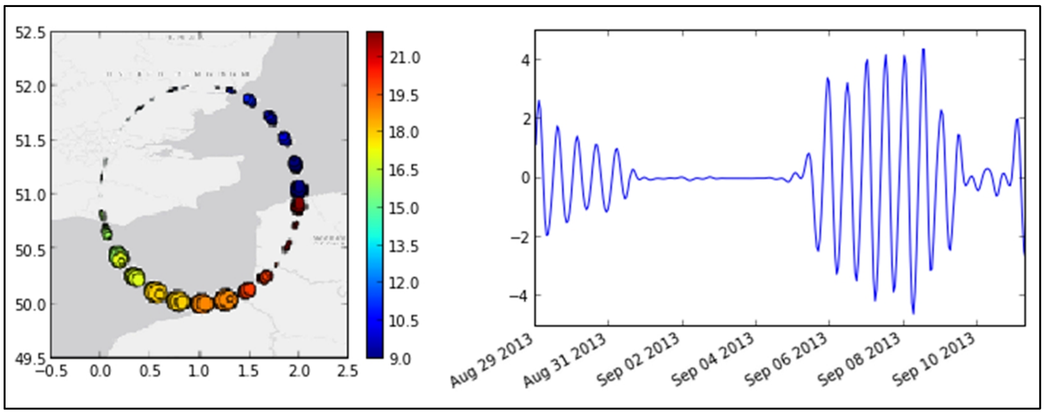


Figure 4.11 - Circular 'sailing' track starting at 51N, 1E (longitude, latitude) and a radius of one degree. The sizes of the dots indicate the water level and the colour of the dots indicates the moment in time (left). The graph on the right shows the water levels observed during the journey of 320 hours (just over 13 days) [GMT]. The track includes parts of the British and French mainland; onshore values are mainly caused by limited resolution of the dataset.

The above examples show there are lots of possibilities of using and improving the basic tool, still other versions of the tool can be made up. However, these are not part of this project. Many users will probably be satisfied with a graphical output of the forecasted water levels. However, for model integration images are not easily interpreted. Users who like to use the tool’s output as input for their model, or for integration of the tool in a modelling environment, graphical data, such as the graphs displayed above, are not practical. For that reason another tool was created that saves the model output to a text file which can be used as model input. The text file type is chosen over other types because this format can easily be made and inspected. Also the format is easy to use as tabular like input for modelling environments. The resolution in time can be adjusted to the user’s requirements, from a forecast each second up to what is practically applicable, for example hours.

5. Case 2 – Interactive Dredge Planning Tool

The second case, described in this chapter, disseminates an existing processing service via WPS by means of a wrapper process. The tool represents the 'Interactive Dredge Planning Tool' (IDPT) which is currently available via the OpenEarth viewer website (viewer.openearth.nl). This tool can be used to forecast the ecological effects of a dredging event in the Singapore area. The choice for disseminating specifically this tool is to demonstrate the applicability of the WPS standard to existing remote processing services.

5.1 Purpose of IDPT

Dredging operations are known to be influencing the environment in a negative way. Fine sediments which are stirred up and returned in the water column due to so called overflowing operations can spread and stay in a state of suspension for quite a long period in time.

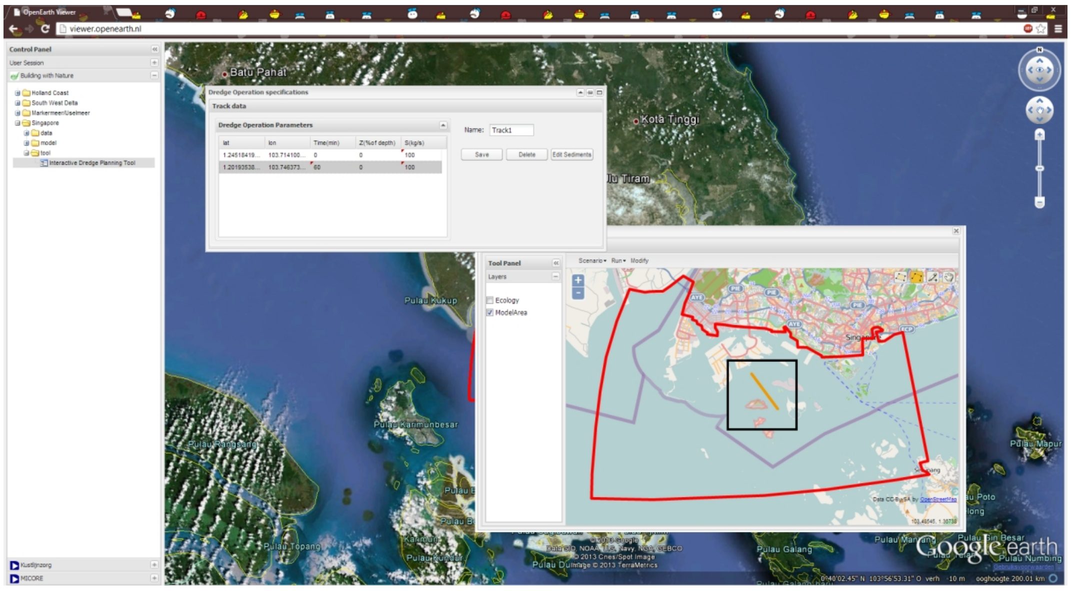
Due to these suspended sediments many negative environmental effects can occur. For example, faster increase of water temperature and coverage of benthic organisms. For these last one should think of for example corals which are usually stationary colonies on a reef, covering these with fine sediments reduces the amount of light that can reach the coral colony and also the amount of nutrients that can be taken up will be limited.

In the Building with Nature (BwN) project several cases on cooperating with nature in coastal engineering were brought in to practise. Besides, a lot of research was performed on the ecological effects of dredging operations and the possibilities to limit these. One of the results of these studies was a tool that was able to forecast the ecological effects of a dredging operation, this is the IDPT.

The tool allows the user to make a complete planning of the dredging activities for the Singapore area, this area is where the model was developed for (modelling extend). As a result the tool shows a map where turbidity is expected due to dredging operations. Also vulnerable areas are assessed to find out to what extent the dredging operations have a negative influence. Depending on the outcome the user can adapt for example the dredge track or the amount of overflowing to reduce the environmental impact of the total dredging operation (Ecoshape, 2012).

5.2 Using IDPT as Original Service

The OpenEarth viewer contains the Google Globe using the Google Earth API (Application Programming Interface). Using this globe the user can specify the location of start and ending points of different tools and models. Using the IDPT offers a lot of options for parameter inputs such as a dredge track, sediment composition and time period. The model is bound to a specific model area where ecologically sensitive areas are specified. Figure 5.1 gives an example of how the graphical user interface of the IDPT looks like.



Dredge Operation specifications

Track data

Dredge Operation Parameters

lat	lon	Time(min)	Z(%of depth)	S(kg/s)
1.24518419...	103.714100...	0	0	100
1.20193538...	103.746373...	60	0	100

Name: Track1

Save Delete Edit Sediments

Figure 5.1 - Graphical user interface of the IDPT in the OpenEarth Viewer. Background shows the Google Earth Globe, the small map window is used for drawing the input path of the dredge track (small orange line in the black circle), the red area shows the modelling extend. The other window is used for other input parameters.

After specifying all input parameters the tool can be run. During the run several status updates are returned to keep the user informed during the processing phase. When processing is finished the result is loaded in the Google Earth API and a reference ID is provided which can be used for calling the specific result in the future. An example of the output is given in Figure 5.2

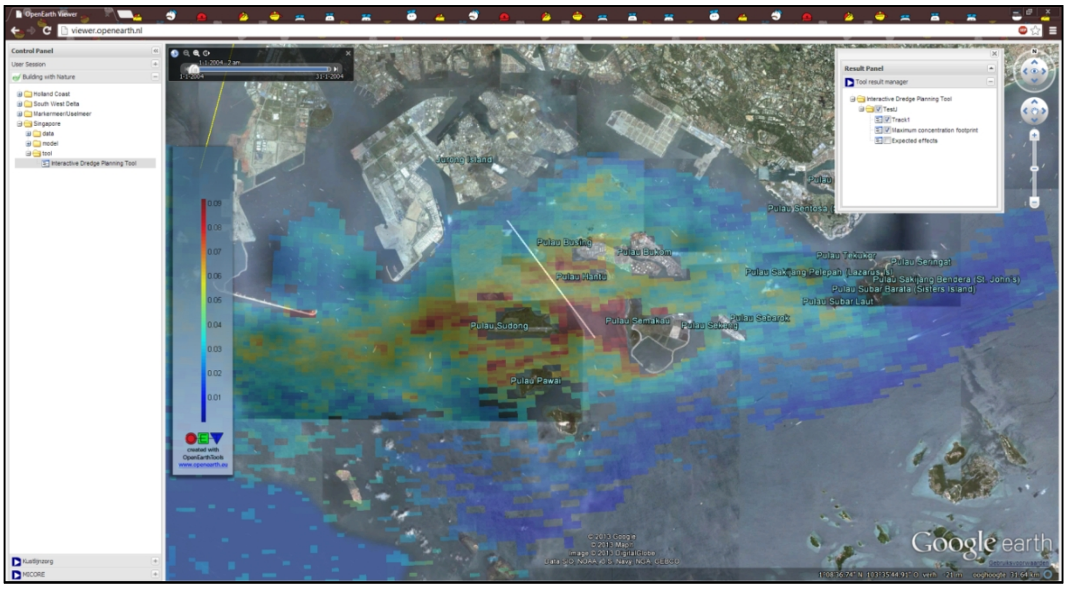


Figure 5.2 - Output window of the IDPT, the input track (white line) and calculated result (blue (low concentrations) to red (high concentrations) colours) are projected on the Google Earth Globe. The window on the right is used for switching different layers on and off.

5.3 Method for Wrapping IDPT

In coastal and marine engineering projects very often Matlab based models and tools are used. Contrary to Python, Matlab is not open source and can therefore be expensive to use. However, within Deltares the number of Matlab scripts and users are abound. Most programmers are usually not very eager of learning a new programming language or, when somebody knows both languages (e.g. Python and Matlab), they are neither very eager of 'translating' existing scripts.

For those reasons it would be interesting to find out whether there is a possibility of running Matlab tools as WPS. Since the WPS server implementation uses Python scripts for its services, a method should be found to make it possible to let the server understand or execute scripts written in Matlab.

Probably the easiest solution is to use a Matlab wrapper which is suitable for Python. A wrapper is a function of an application which is able to call or run a different application. The wrapper that is available for executing Matlab code from Python is Mlabwrap. Mlabwrap is a Python library can be used as such. By importing the Mlabwrap library to the Python script Matlab functions can be called. To get the library working Matlab should still be installed on the computer or server. Unfortunately this is not possible at Deltares since the Matlab license server is behind a firewall.

Due to the situation of the technical installation at Deltares the option above cannot be used for this project. Although there is a Matlab server available, the server used as WPS server does not have Matlab installed.

For using Matlab scripts from the WPS an instruction will be sent from the WPS server to the Matlab server which on his turn should return the result in an understandable format. This will result in a route (sequence diagram) for instructions such as in Figure 5.3. The PyWPS WPS service and the Matlab server communicate using files. The WPS for example can generate and instruction file (e.g. m- or XML file) for the Matlab server which on its turn waits for an instruction to be posted. Since the WPS standard also allows for status updates during the processing phase, and update can for example be given when the Matlab server starts executing the process.

For this project the Python script of the WPS is built as such that it is completely compliant to the way the Matlab script is working. For future projects a convention is proposed (internally at Deltares), for setting up Matlab scripts that will be disseminated as WPS (Baart, 2013). Keeping up to this scripting style will result in easy adoption and integration of Matlab tools as WPS.

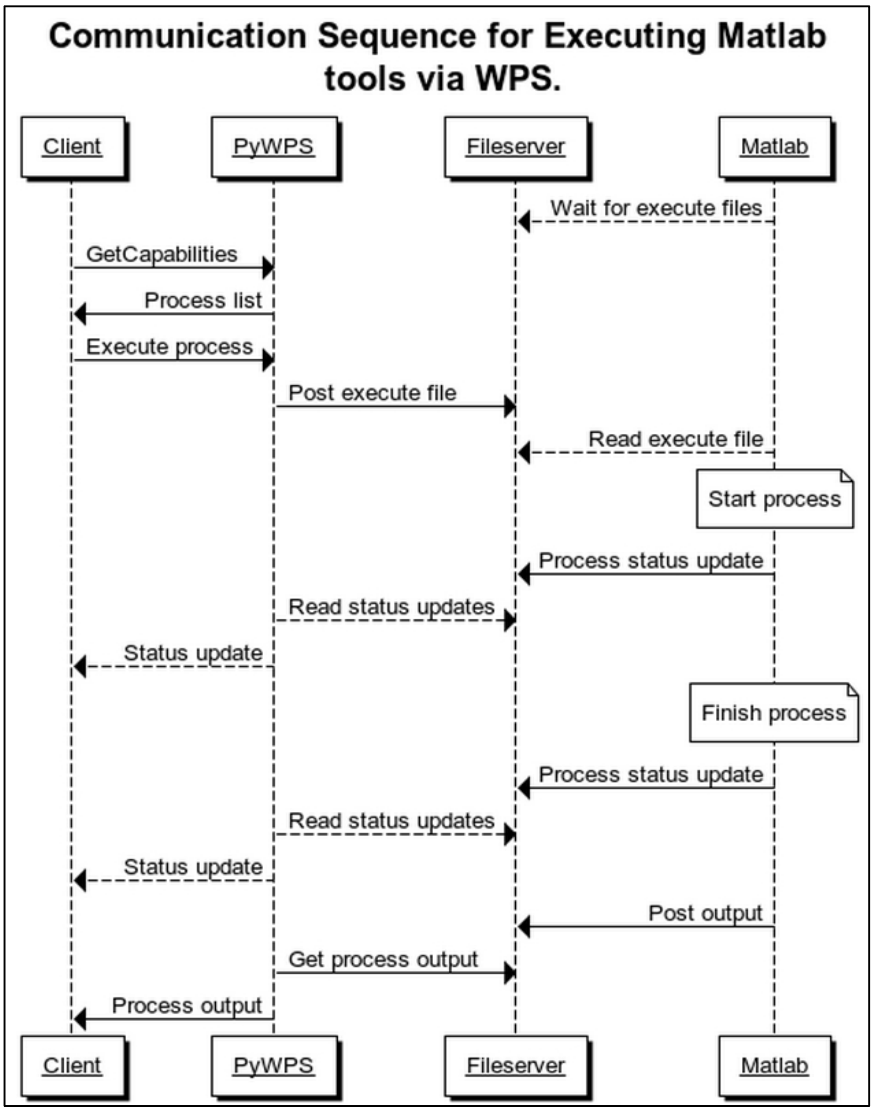


Figure 5.3 - Sequence diagram for executing Matlab tools using WPS via an external (model) server. The fileserver (FTP) is used as a bridge between the internet (client and PyWPS) and local network (Matlab).

5.4 Wrapping IDPT as WPS

The advantage of the current setup of the IDPT is that there is already a backend interface which can work independently of the graphical user interface. For this reason the processing service only has to be able to create and post an XML file to the server, read the status file and download the output file(s).

The model offers a quite large range of input parameters and options which can become quite extensive to wrap for 'just' a proof of concept. For that reason a stripped down version of the model is offered as WPS. In this version only a two-point vector (single line) can be used as an input parameter. Other parameters that can be changed are the number of particles and the period in time.

The original service uses XML files to communicate with the model using a FTP connection. For that reason the WPS process also needs to be able to connect to the FTP and save and read files from there. Below the reading and writing cycle is described according to Figure 5.4.

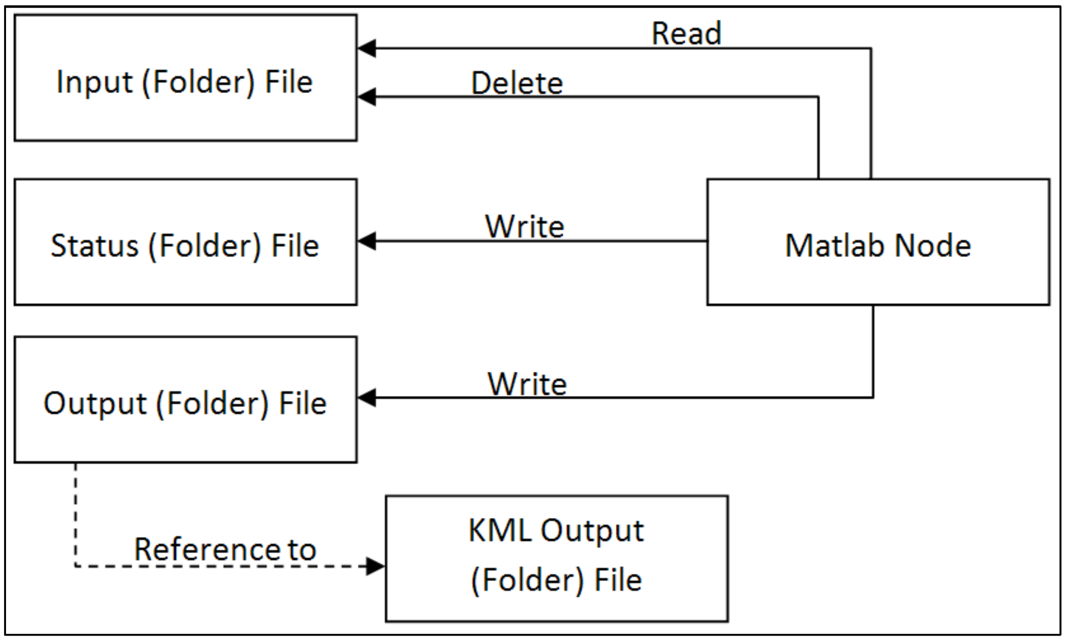


Figure 5.4 - Schematic view of the reading and writing cycle of the IDPT process.

The tool generates a file for every step in the process which is written in a different folder. The first step in the process is to create and save an input file in the 'input' folder. That file is used to start the model run. After executing, the model creates a status file in the 'status' folder and removes the input file. In the wrapper, the input file is built using a Python script from PyWPS, the input URL containing allowable parameters can be found in the Execute request in Figure 5.5.

```
http://dtvirt5.deltares.nl/wps/?
      request=Execute&
      service=wps&
      identifier=IDT_simple&
      version=1.0.0&
      DataInputs=[
             lat1In=1.21462;
             lat2In=1.24757;
             lon1In=103.687;
             lon2In=103.728;
             scnameIn="testJ";
             nparticles=10000;
             tdurationIn=1
             ]
```

Figure 5.5 - Execute URL of IDPT, input parameters are locations of point one (lat1In and lon1In) and two (lat2In and lon2In) (input vector), the scene name (scnameIn), the number of particles (nparticles) to base the calculation on and the duration of the scene (tdurationIn).

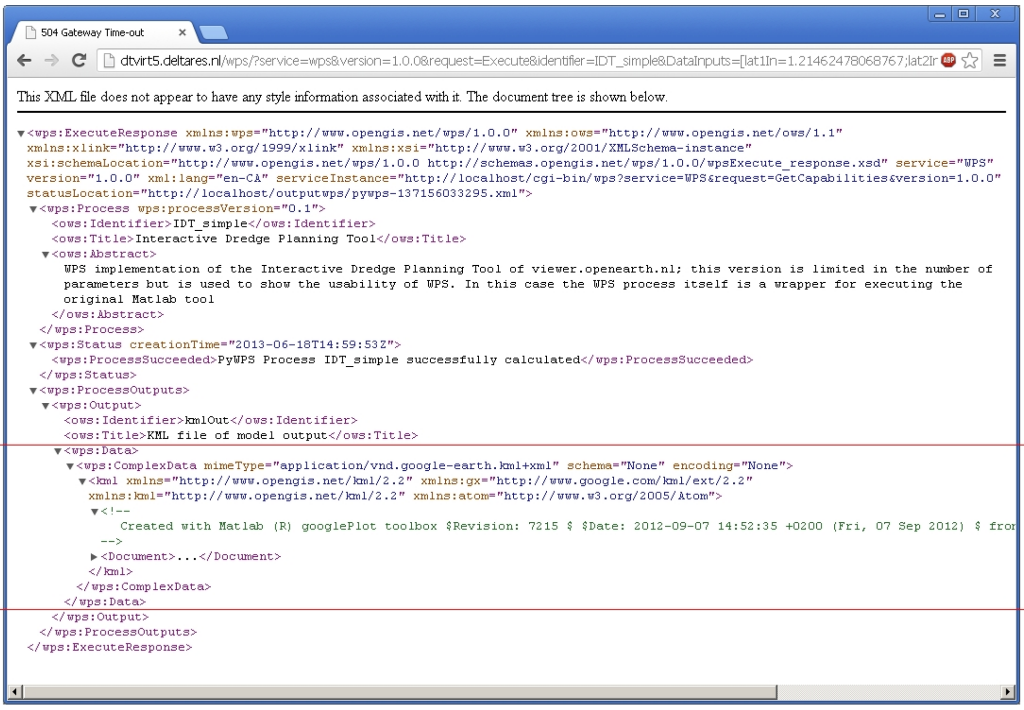
To prevent from overwriting old statuses and output files, each model run receives a random five character code for identification. This code is part of the file names of the input, status and output. For these files, the code is identical which makes it relatively easy to guide the wrapper to the correct file to read. To mimic the code generation and prevent from overwriting files, the WPS process reads all filenames in the status folder and generates a non-existing code. This is actually a more safe method than the original IDPT interface which suggests that a random code is unique enough to prevent from overwriting.

The Matlab Node (on which the IDPT is running) continuously reads the input folder to check whether a new processing request is available. If so, a status file will be generated and saved in the status folder. While running the process, the status file is overwritten when a status update is available. For this reason the WPS process continuously reads the status file and updates the status when a new one is available. It could be possible that a certain status update is missed by the WPS when an update is actually given. This is however not a problem since the last status update (which says the output file is ready) is not updated or removed anymore. This last update is the only one that triggers the WPS to continue in the process. The script for reading and publishing the status is in Figure 5.6.

```
import time
time.sleep(3) #Time required for the ftp to store the file
server = ftplib.FTP(host=host, user=user, passwd=passwd)
server.cwd(status)
status = 'start'
a = 0
status_old = status
while status != 'Uploading output file':
    output = cStringIO.StringIO()
    statfile = server.retrlines("RETR {}".format('status_'+uID+'.xml'), output.write)
    output.seek(0)
    s = lxml.etree.parse(output)
    status = lxml.etree.tostring(s.find("status"))[8:-9]
    if status == 'Uploading output file':
        self.status.set(status, 95)
        self.status.set("Ready to access output files", 100)
        break
    if status != status_old:
        self.status.set(status, a)
        a = a + 5
    status_old = status
```

Figure 5.6 - Script used to read the current status. The script reads the status XML file generated by IDPT. The status is read and printed as WPS status, as soon as the IDPT status is 'Uploading output file' the loop breaks and the process continues to the next step, finding and reading the output files.

As soon as the status is updated in 'Uploading output file', the model output can be accessed. The Matlab Node now writes a file in the output folder which contains the name of the output file. This filename contains a new five character code so it is not possible to directly access the files in the KML folder using the previous code. This action is caused by the original (programming of the) tool, the reason for generating this new code is not clear. To speed up the process it would be more optimal to use the same code as the previous because the output folder can then be accessed directly. When the filename is read from the output XML, the ZIP file can be accessed in the KML folder. This ZIP file contains three different output files from which one is the requested KML file. This KML file is unzipped and printed in the WPS's output XML, see Figure 5.7.


504 Gateway Time-out

dtvirt5.deltares.nl/wps/?service=wps&version=1.0.0&request=Execute&identifier=IDT_simple&DataInputs=[lat1In=1.21462478068767;lat2Ir

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0" xmlns:ows="http://www.opengis.net/ows/1.1"
 xmlns:xlink="http://www.w3.org/1999/xlink" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.opengis.net/wps/1.0.0 http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
 version="1.0.0" xml:lang="en-CA" serviceInstance="http://localhost/cgi-bin/wps?service=WPS&request=GetCapabilities&version=1.0.0"
 statusLocation="http://localhost/outputwps/pywps-137156033295.xml">
  ▼<wps:Process wps:processVersion="0.1">
     <ows:Identifier>IDT_simple</ows:Identifier>
     <ows:Title>Interactive Dredge Planning Tool</ows:Title>
    ▼<ows:Abstract>
       WPS implementation of the Interactive Dredge Planning Tool of viewer.openearth.nl; this version is limited in the number of
       parameters but is used to show the usability of WPS. In this case the WPS process itself is a wrapper for executing the
       original Matlab tool
     </ows:Abstract>
   </wps:Process>
  ▼<wps:Status creationTime="2013-06-18T14:59:53Z">
     <wps:ProcessSucceeded>PyWPS Process IDT_simple successfully calculated</wps:ProcessSucceeded>
   </wps:Status>
  ▼<wps:ProcessOutputs>
    ▼<wps:Output>
       <ows:Identifier>kmlOut</ows:Identifier>
       <ows:Title>KML file of model output</ows:Title>
      ▼<wps:Data>
        ▼<wps:ComplexData mimeType="application/vnd.google-earth.kml+xml" schema="None" encoding="None">
          ▼<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2"
           xmlns:kml="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">
            ▼<!--
                Created with Matlab (R) googlePlot toolbox $Revision: 7215 $ $Date: 2012-09-07 14:52:35 +0200 (Fri, 07 Sep 2012) $ fror
             -->
            ▶<Document>...</Document>
           </kml>
         </wps:ComplexData>
       </wps:Data>
     </wps:Output>
   </wps:ProcessOutputs>
 </wps:ExecuteResponse>
```

Figure 5.7 - XML output of IDPT. The output KML is written in between the ‘wps:ComplexData’ tags (for clarity reasons the KML output is collapsed).

6. Discussion

In this chapter findings on the results are given. These reflect on both the WPS standard and the used software (PyWPS) with implemented services from the test cases.

6.1 PyWPS

PyWPS offers a relatively easy way to offer WPS compliant Web Processing Services. It offers the possibility to create any kind of process as long as it fits in the Python language. One of the main issues relevant for this project is the updates of statuses, this is discussed under the IDPT test case (section 6.2.2).

The community behind PyWPS is relatively small, two developers and seven contributors (Garnett and Fenoy, 2011). Only a limited number of people have actually been contributing to the development of PyWPS. A (possible) result of this is that not all aspects are developed and available as much as one would like. Especially the documentation of PyWPS is limited, for users new to the system this can be a great drawback. However, in the project we got PyWPS running locally and on the web server without limitations this in contradiction to the 52North software.

Due to the limited number of contributors, development is also relatively slow. The used version of PyWPS in this project is 3.2; this version has been available since June 2011. Just recently, early June 2013, a new version, PyWPS 4.0 has been taken in development. The main difference with the previous version is that it will support Python 3.

Just to show a difference with other development teams, 52North, creator of 52NorthWPS has about 25 active contributors for development (Garnett and Fenoy, 2011) and is supported by different companies and research institutes. Also a relatively extensive documentation is available and updates are more regularly (about 4 times per year) launched.

6.2 Test Cases

Both test cases are used for demonstrational use. Both services give proper results, the tide forecasting service was checked with actual water levels in the Dutch North Sea and the IDPT service gave the same output as the original tool. However, especially for the tide forecasting tool accuracy is not optimal and results should not be used in the same way as the ‘regular’ forecasts which are expected to be more reliable. The limited accuracy is caused by the resolution and limited validation of the used dataset. However, the shape of the tidal curve matches actual measurements so moments of high and low water can be forecasted with the service.

In both test cases some issues were found in the use and application of the tools. Some can be imputed to the system setup while others can be referred to the human errors or limitations in the existing wrapped tool.

6.2.1 Issues Found in the Forecasting Service

The practical work on the tide forecasting tool was performed without many issues. The actual issues that came up are already discussed in 4.4.2. There issues were mainly related to a shift in time while comparing the tool's results with actual measurements due to daylight savings and finding the right starting point in time of the TPXO dataset. In the forecasting service no issues were found related to the WPS standard.

The time issue can be solved by creating some extra code in the script where the user can define from which time zone the starting time is given. The Python function for this feature is already available and can be added to the tool when preferred. The disadvantage of adding a time zone is that an additional input parameter has to be added to the already extensive Execute request URL. In manually using a WPS certain request URL's can be seen as an issue in usability. The use of for example the QGIS plug-in will limit this issue.

6.2.2 Issues Found in Wrapping IDPT

During the practical work on the IDPT wrapper some issues were found. Most of these issues are solved, otherwise solutions are proposed.

Statuses

In the IDPT wrapper the statuses from the actual tool are read and published. Receiving status updates can help the user to keep his patience. In the actual tool the user receives the status updates from a window in the user interface. PyWPS should also be capable of returning the process' status to the client; however this does not seem to be working correctly. The status function is implemented correctly and is recorded in the server's log files. However, the status is not returned to the client, not when using a browser and neither when using the QGIS plug-in. After some checks in the source code of different components it seemed that the problem originates from Gunicorn, web server software, unfortunately this problem is still unsolved.

Timeouts

The IDPT takes quite some time to do a complete run; this can be experienced in the original interface. Wrapping the function does not make the process any faster, the actual processing time is about 5 seconds longer using the wrapper. This increase in processing time is not related to the use of WPS but due to the communication method (via a FTP server) of IDPT.

A risk in this situation is that there are several components in the setup that can give a timeout error. When testing the wrapper it was found that this indeed happened. A timeout can be given by the used browser (Google Chrome), server (NGINX and Gunicorn) or server instance (PyWPS). It was found that the timeout was raised by Gunicorn, the WSGI server software. After setting the timeout period to 120 seconds the problem was solved. When statuses are published to the client timeouts can be prevented. Increasing the timeout period is not a very strong solution, especially not when processes with longer running times are implemented. When status updates prevents from timeouts a solution could be to return, for example, every ten seconds a new update. The message given by the status is off less importance in that case.

When setting up more extensive processes which take a longer period to process, one should consider what time should be set for the timeout (on all components) or find a different method for returning the result. Regular web users do not like waiting for results, four seconds is seems to be too long (Lohr, 2012). It is expected that people who perform model runs professionally are a bit more patient and probably used to longer running times. However, when the run takes over several minutes it could be better that the user is informed (for example by email) when the process is completed. For this kind of processes, methods by Zhao et al. (2010) for asynchronous processing could be proposed.

Time Consuming FTP Connection

The first issue found in wrapping IDPT was the issue of accessing status and output files of the tool. While testing the wrapper, an error about missing files raised. It seemed to be that the files that were written as the tool's status and output were not ready yet when the wrapper actually was. Unfortunately, storing the files via the FTP connection seems quite time consuming, for that reason after each FTP file storing operation a three second break is built in the script. Due to this break the service is just a few seconds slower compared to the original IDPT.

6.2.3 QGIS integration

For testing purposes of the services the QGIS plug-in and a web browser are used. In the browser, the full XML output file can be read but seems somewhat less user-friendly and more sensitive to typing errors. The QGIS plug-in reads which input parameters are required for the process and shows these in the request form that can be filled out by the user.

As an output the plug-in is expected to recognize the output type, in this case KML, and loads it as a layer in the workspace. However, the plug-in does not seem to recognize the KML file and does not show any result or errors. A search in some example scripts show that the plug-in only reads GML or plain text output from the WPS's XML file. Using the Python OGR library (Python GIS module) the tool's output could be converted to GML. In future versions the plug-in might as well be able to read files different from GML. It is not known whether other GIS software packages support outputs different than GML. A solution is to chain processes and do a file conversion in the last process. In the IDPT case, the chain would exist of two steps, the wrapper and a conversion tool to transform the KML output file in GML format.

6.2.4 Performance

Both services take quite some time to process. For the IDPT there is not much that can be done to make improvements. The WPS version of IDPT takes only a few seconds longer than the original version. This increase of processing time is mainly due to the built-in waiting time for the FTP server to write the files.

The tide forecasting tool takes about 15 seconds to process an Execute request and provide an output. The most time consuming part of the service is loading the TPXO dataset and do preparing calculations on it. The dataset has a file size of about 400MB which is re-read with every request. When accessing this file from another (OPeNDAP) server, processing time increases even more.

There are some options for performance improvement. A first one that can be thought of is splitting (tiling) the file in different areas on the earth. Depending on the requested location the required partial data set can be loaded. When splitting for example in 8 parts, the dataset only has a size of about 50MB.

Another dataset which can be used is version 8 of TPXO. Usually this dataset is way larger compared to the 7.2 version that was used in this project. However, within the organization responsible for the TPXO dataset, Oregon State University, researchers have been working on reducing the size of the file(s) by converting these to a newer version of NetCDF, the total size for this converted version is about 75MB. Not only is it expected that using this dataset offers better performance, also accuracy will improve due to the higher resolution of the dataset.

Improvements can also be made in the script. Currently the data file is loaded and directly an interpolation is made over the entire dataset. The script can be adapted that is only interpolates a small area based on the requested locations. It is expected that this would reduce processing time a lot.

7. Conclusions and Recommendations

As a result of the literature study and cases, this chapter gives conclusions and recommendations based on the research questions.

7.1 Conclusions

1. Can complex processes that feed coastal/marine models be exposed via a Web Processing Service?

Yes, complex processes which feed coastal/marine models can be exposed via a WPS. WPS is a (very) general standard that can be applied to about any type of process or data. The standard prescribes the way that processes are called (i.e. capabilities, describe and execute) and how the in- and outputs are communicated to the client (XML). Supported by the results of the test cases it can be said that complex processes, independently of a domain, can be exposed via a Web Processing Service.

Currently operational limitations of the services are caused by the server implementation, for the test cases this is PyWPS. PyWPS supports the Python language for processing scripts, although Python is gaining more and more popularity in many fields of work, other languages are used as well. Besides this, PyWPS offers a rather static implementation of the available processes. For example It is not possible to select a range of processes depending on a certain input. It also lacks a number of possibilities, for example to return a model extent (complex output) in the DescribeProcess output.

The test case of the Tide Forecasting tool shows that some effort has to be done before a suitable tool can be created. The main disadvantage is that some tools that are available, do not easily fit in a (Py)WPS script. This can be because of the use of a different programming language (e.g. MatLab) or that the program is not yet practically operational (e.g. Tappy, a Python program that unfortunately cannot be used as a library by other Python scripts). Different than planned at the start of the project, a forecasting tool also had to be made, this raises the question whether a global forecasting tool could be programmed.

A global tide forecasting tool can be created based on satellite measurements. In this project the tool uses a processed dataset of TOPEX/Poseidon data which covers the entire globe. With the use of Python these data points can be processed such that the amplitude of all available tidal constituents can be calculated at any moment in time. Because the dataset has a limited resolution (0.25 degree) a simple interpolation operation is performed. As a result, the tool shows that it is possible to create a global tidal forecasting tool.

Dissemination of the forecasting tool via WPS is relatively simple. The tool is programmed in IPython as a ‘stand-alone’ working script so adapting it to fit the PyWPS process layout did not take too much effort afterwards. Operations for adapting the script mainly exist of defining the in- and output parameters in the PyWPS format.

2. *Can a Web Processing Service be used as a wrapper service for non domain-specific processing services?*

From the perspective of the WPS standard it is possible to create about any process the users is capable of programming, so a wrapper is also in the range of possibilities. As shown by the IDPT wrapper test case it is actually possible to do so. In this case a Matlab processing service was called via a WPS process. One should however keep in mind that the working of non standardized processing services can be quite intensive to figure out. When the programmer of the original service knows his program will be made available via WPS in the (near) future, the program could be written in such a way that that implementation of a WPS wrapper service is relatively simple. This can be done for example by programming according to prescribed conventions. Such a convention (for Matlab) has recently been proposed within Deltares.

3. *Which adaptations could be made to the WPS standard to improve the possibilities for application of the standard?*

As mentioned before, the standard itself is very general and can be applied for about any kind of process. The main issues found are related to the implementation of the standard in software packages. However, some findings in this project give room for possible changes in the actual standard.

Ideas for improvements are a result of the long running time of the IDPT test case. One of the issues found was a time out of the server during the execution of the tool. An improvement in this case could be to dynamically sent updates of the status of the process or at least the time the process has been, or will be running. In this way there will be continuous communication between the client and the server which prevents the server (but also the client) of giving time outs.

Another suggestion for improving computationally intensive models, which for example require a longer running time than IDPT (about 2 minutes), is to use an asynchronous service. In this way the server informs the client when the process is finished and if so, where the output can be found. When using such a processing method there should be an option for the client to request the status of the process (e.g. how much time is left before the process is finished or if there have been any errors during the process execution). Also the possibility to stop the running process would be an interesting option in such a situation. When a process is expected not to finish in time or the process crashes or hangs because a certain dataset loads very slowly there should be an option to stop the process from running to save server time.

A final remark on the WPS standard is that it limits options for dynamic process interfaces. For example the IDPT tool can only operate in a certain area. When having a completely functioning WPS version of this tool, this model boundary should for example be returned in the output of the 'DescribeProcess' request, in the WPS specification however there is no room for such 'complex output' in this request.

Also tools could be coupled when after an initiation phase additional parameters have to be added or changed. For example, in a certain model the client needs to provide the grain size of sediment being modelled, depending on this grain size other parameters are calculated. However, when the client wants to check and possibly adapt the calculated parameters the model should exist of multiple processes which are chained in some way. It would be nice if parameters could be given as input in different stages of the processing operations.

Remote processing requires data shipping; this means that data is send to a processing server. Depending on the size of the dataset it can be less time consuming to execute the process at the location of the data, this is called process shipping.

Overall Conclusion

The WPS standard is able to handle many types of processes. In the test cases it is shown that both a self programmed process (tide forecasting tool) and wrappers for other remote processes can be handled by the WPS standard. Although WPS is intended as a standard for geoprocessing, also processes without any kind of geographical component can be disseminated via the WPS standard.

To improve the workability of long running processes some improvements are suggested. A solution should be found for time outs, process statuses should be requestable and it should be possible to stop processes during execution. Based on experience gained from the test cases it may feel like WPS does not offer fast responses from processes. This is however far from true, WPS will not speed up existing processes but in the test cases there can be much improved by optimizing the process' script of the Tidal Forecasting Service. In the IDPT case the tool itself already takes quite some time to run, because the WPS wrapper runs exactly the same tool it is impossible to reduce processing time.

The WPS standard only allows for somewhat static processes. Processes can become more dynamic by coupling different processes or adding complex outputs for GetCapabilities and DescribeProcess requests. However, no options are available for adding new or different parameters once the process has started. However, when all required parameters are available beforehand coupling would only be necessary to link different 'sub-processes' which on the other hand could also be deployed in a single process.

Overall it can be concluded that WPS offers great options for interoperability of different systems for many fields of science. Still there is room for improvement and extension of the abilities of the standard however, some have already been proposed to take into account for a future version.

7.2 Recommendations

Based on the experiences and findings of this project some additional recommendations are made. Note that in the previous section some recommendations are already given specifically for the WPS standard.

7.2.1 Tide Forecasting Tool

In its current state the tide forecasting tool operates as expected. However, some improvements can be made for better performance, accuracy and workability.

Performance, in terms of processing time, of the tool itself is not very good, improvements could (or should) be made. When running the script locally in a Python editor, the first run takes about as long as a run on the server (about 15 seconds) while following runs in the editor show much faster results (about 2 seconds). The main performance issue can probably relates to loading of the dataset and not optimally programmed operations (e.g. interpolating operations on the entire dataset instead of the requested area or point). The used dataset has a size of about 400MB which is re-loaded and calculated for every request. However, the actual calculating process for water levels is relatively fast. It is recommended to make use of tiling or make the server cache the data at the first run so following requests are expected to perform like the requests in the Python editor.

The results given by the tool are not validated to other forecasting models. However, a check with actual water levels and shape of the tidal curve at a measuring platform at the Dutch coast showed that the results given by the tool are in line with the actual measurements (shape of the tidal cycle). It is expected though that the results are not optimal, for that reason it is recommended only to use the tool for demonstrational purposes.

A first step in improving the tool is to use a newer, higher resolution dataset. This dataset is already available; TPXO 8; This dataset has not been used due to the even larger file size. However, it does offer higher resolution and values were validated with actual measurement stations.

The input parameters of the current tool are all character based. Workability could be improved when also ‘graphical’ input would be allowed as location input. The QGIS plug-in for example allows users to use points, vectors or polygons as input for WPS based services. The input type should then allow the GML type. Also this GUI like possibility can be used to visualize output data, for example by returning values for water levels on a vector.

7.2.2 IDPT Wrapper

The IDPT wrapper is now used for demonstrational purposes. The functionality of the wrapper is limited compared to the actual tool, this functionality should be improved when the IDPT will officially be disseminated via WPS.

Extensions should be implemented in the number of input parameters. This number is currently the main limiting factor. Some of these parameters are the result of sub processes of the actual tool. If these are also implemented, the wrapper should chain multiple processes from one script.

The actual IDPT uses a practical graphical interface where users can manually draw multiple vectors. Such possibilities should also become available for using the tool via WPS. In the current WPS version it is only possible to add a single vector by manually giving in the coordinates of the start and end points of the dredge track. Extending the possibilities of giving in points, vectors or polygons in a graphical user interface, for example by using the QGIS plug-in, will make the tool more attractive.

Currently status support during the IDPT processing is not working optimally. During the run of the wrapper the status is published in the interface of the server and in the server log file. Because the processing time of the service is too long to wait without any interaction, statuses should be communicated to the client. It is recommended to make it possible to display the current status of the tool at the client’s screen, whether this is in a web browser or in a dedicated WPS client such as a desktop GIS.

The amount of processing time compared to the original service does not increase much. However, the original service is not completely optimized. For example the use of storing files on an FTP server is quite time consuming. It is recommended to optimize the tool as such that processing time will be reduced. In that way also the processing time of the wrapper can be reduced.

7.2.3 Other Recommendations

The QGIS Plug-in is rather limited for working with WPS services. Although the plug-in offers a simplified and convenient method for operating WPS’s, there are some limitations. The main limitation is that the plug-in only allows GML based output for visualization in QGIS. The only other output format somewhat supported is plain text (based). The result, when text, is shown in a popup window after processing is finished. This result however, cannot be saved.

Extension of the number of allowed file types may help in extending the number of users. In the IDPT service, the output file is a KML file printed within the XML output file. Although this file type is supported by QGIS, it seems like the plug-in refuses to do anything with the file, not even it is displayed as text. Because a lot of visualizations of Deltares are in KML format it is not very likely that the plug-in will be used a lot within Deltares when Deltares WPS’s offer KML as (preferred) output format.

To use WPS in real world modelling environments, the modelling software has to be adjusted such that it can communicate with WPS servers. The advantage of allowing a WPS as input is that any tool disseminated via WPS can be used as input. Not only marine processes but also already available GIS tools can be used. Besides that, after implementing a WPS it is only a small step to also allow other web services, such as maps and other data files as model input.

The implementations of the two case studies show that the marine field can benefit from the emerging GI standards. The difficulty in applying such standards is often not a technical challenge but finding the intersection of the two jargons.

Providing models as services allows the person who creates a service to focus on the core area of interest (for example predicting tide). Presenting the service in a user friendly interface can be left to developers of desktop programs (ArcGIS, QGIS and model setup tools such as Deltares' Deltashell) or to web developers (for example OpenLayers).

8. References

52North, (2013) Using Python Scripts as WPS Processes. Retrieved March 2013 from 52North Geoprocessing Python Tutorial:http://52north.org/communities/geoprocessing/wps/tutorials/python_ags_tutorials/python_tutorial.html

Baart, F. (2013) Matlab WPS Convention. Retrieved May 2013 from Deltares Public Wiki. Version of April 19, 2013. https://publicwiki.deltares.nl/display/OET/Matlab+WPS+convention.

Baranski, B., Deelmann, T., Schäffer, B. (2010) Pay-per-Use Revenue Models for Geoprocessing Services in the Cloud. ISPRS Proceedings of the 1st International Workshop on Pervasive Web Mapping, Geoprocessing and Services, August 26-27, 2010, Como, Italy.

Bosboom, J., & Stive, M. J. (2011). Coastal Dynamics I: Lecture Notes CT4305. VSSD.

Castronova, A.M., Goodall, J.L., Elag, M.M. (2013) Models as web services using the Open Geospatial Consortium (OGC) Web Processing Service (WPS) standard. Environmental Modelling & Software 41, pp. 72-83.

Diviacco, P. (2005) An open source, web based, simple solution for seismic data dissemination and collaborative research. Computers and Geosciences 31, pp. 599-605.

Dubois, G., Schulz, M., Skøien, J., Bastin, L., Peedell, S. (2013) eHabitat, a multi-purpose Web Processing Service for ecological modelling. Environmental Modelling & Software 41, pp. 123-133.

Ecoshape (2012), Interactive Dredge Planning Tool Singapore. Retrieved online from https://publicwiki.deltares.nl/display/BWN/Tool+-+Interactive+Dredge+Planning+Tool+Singapore. June 2013.

Egbert. G.D., and S. Y. Erofeeva. (2002) Efficient inverse modeling of barotropic ocean tides. Journal of Atmospheric and Oceanographic. Technology, 19, pp. 183-204.

Farres, J., (2009), G-POD, ESA Grid Processing on Demand for Working Scientists. Presentation at ESRIN November 13, 2009.

Gallagher, J., Potter, N., Sgouros, T., Hankin, S., & Flierl, G. (2004). The Data Access Protocol—DAP 2.0. ESE-RFC 004.0. 03 Category: Proposed Community Standard, 2004/09/14 (NASA Standard Process Group webpage: http://spg. gsfc. nasa. gov).

Garnett, J., Fenoy, G. (2011, September 15) WPS Shootout. Tech Session presentation at FOSS4G Conference in Denver 2011.

Jesus, J. de, Dubois, G., Hiemstra. P. (2008) Web-based geostatistics using WPS. GI-Days 2008, Münster, Germany.

Kalin, M. (2009) Java Web Services: Up and Running. Beijing: O'Reilly.

Lohr, S. (2012) For Impatient Web Users, and Eye Blink Is Just Too Long to Wait. The New York Times (Published: February 29, 2012) Retrieved june 12 from http://www.nytimes.com/2012/03/01/technology/impatient-web-users-flee-slow-loading-sites.html.

Lopez-Pellicer, F.J., Rentería-Agualimpia, W., Béjar, R., Muro-Medrano, P.R. (2012) Availability of the OGC geoprocessing standard: March 2011 reality check. Computers & Geosciences 47, pp. 13-19.

Michael, C., Ames, D.P. (2007) Evaluation of the OGC Web Processing Service for Use in a Client-Side GIS. OSGeo Journal Vol. 1, May 2007 pp. 1-8.

Morozov, I., Reilkoff, B., Chubak, G. (2006) A generalized web service model for geophysical data processing and modelling. Computers and Geosciences 32, pp. 1403-1410.

Nativi, S., Mazzetti, P., & Geller, G. N. (2013). Environmental model access and interoperability: The GEO Model Web initiative.Environmental Modelling & Software,39, 214-228.

NASA (2006), NASA's Topex/Poseidon Oceanography Mission Ends. NASA News Releases 06-001, online: http://www.nasa.gov/home/hqnews/2006/jan/HQ_06001_TOPEX_update.html.

NASA (2012), NEX Facts. Published October 26, 2012. Retrieved online from the NEX website https://c3.nasa.gov/nex/projects/1304/ on July 2, 2013.

Neteler, M., Bowman, M. H., Landa, M., & Metz, M. (2012). GRASS GIS: a multi-purpose Open Source GIS.Environmental Modelling & Software,31, 124-130.

OGC, Schut, P. (2007) OpenGIS Web Processing Service. OGC Standards Document 05-007r7 version 1.0.0.

OGC (2011) OGC Network Common Data Form (NetCDF) Core Encoding Standard version 1.0. OGC Standards document 10-090r3.

OGC (2013) retrieved February 2013 from the OGC History page: http://www.opengeospatial.org/ogc/historylong

Pawlowciz, R., Beardsley, B., Lentz, S. (2002) Classical tidal harmonic analysis including error estimates in MATLAB using T_TIDE*. Computers & Geosciences 28 (2002) 929-937.

Phillips, T. (n.d.) Tides and tide prediction. Retrieved March 2013 from Stony Brook University, http://www.math.sunysb.edu/~tony/

Pinet, P. R. (2006). Invitation to oceanography. 4th ed. Sudbury, Mass.: Jones and Bartlett Publishers.

PyWPS (2011) PyWPS Wiki page. Retrieved March 2013 from http://wiki.rsg.pml.ac.uk/pywps

Steenbergen, E. (2004) Improving the Topex/Poseidon calibration procedure by modelling and implementation of glacial isostatic adjustment. Delft University of Technology, MSc thesis.

Vaughan-Nichols, S.J. (2002) Web Services: Beyond the Hype. IEEE Computer, 35(2):18 21.

Vries, de M. And Oosterom, P. (2008). Model Generalization and Methods for Effective Query Processing and Visualization in a Web Service/Client Architecture. Spatial data on the web, pp. 85-106.

W3C (Network Working Group) (1999) Hyptertext Transfer Protocol – HTTP/1.1. Online: http://www.w3.org/Protocols/rfc2616/rfc2616.html

W3C (Network Working Group) (2007) SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Online: http://www.w3.org/TR/soap12-part1/ (27 April, 2007)

W3C (Network Working Group) (2004) Web Services Glossary. Online: http://www.w3.org/TR/ws-gloss/ (11 February 2004).

W3C (Network Working Group) (2010) XML Essentials. Online: http://www.w3.org/standards/xml/core

Weaver A.C. (2004) Web Services. University of Virginia, Department of Computer Science. Lecture presentation CS651/551.

Weiss, A. (2007). Computing in the clouds. Networker, 11(4).

Wright, J., Colling, A., & Park, D. (1999). Waves, tides and shallow-water processes. 2nd ed. / Oxford: Butterworth-Heinemann, in association with the Open University.

Woodworth, P.L., Player, R. (2003).The permanent service for mean sea level: An update to the 21st century. Journal of Coastal Research, 19-2, pp. 287-295.

Zhao, P. Di, L., Yu, G. (2012) Building asynchronous geospatial processing workflows with web services. Computers & Geosciences 39, pp. 34-41.

9. Appendices

I. Setting up PyWPS in a Windows environment

II. Implementation of WPS Processes

I. Setting up PyWPS in a Windows environment[7]

The PyWPS software was originally developed to use under Linux (Debian). However, it is also possible to use the software in a Windows environment for example for testing purposes. To do so, Apache server has to be installed on the system.

Step 1:

Download PyWPS

Step 2:

Unpack the PyWPS folder and save the entire folder as 'c:\pywps'

Step 3:

Create a folder called c:\pywps_processes and add the file 'default.cfg' (from 'c:\pywps\pywps'), if you like you can insert your own information in the file.

Step 4:

Navigate to the Apache folder. In this folder you find a sub folder called cgi-bin, create a file called pywps.cgi in this folder and insert the following code in it:

```
importsys

sys.path.insert(0, r"c:\pywps")

importos

os.environ['PYWPS_CFG']='C:/pywps_processes/default.cfg'

os.environ['PYWPS_PROCESSES'] ='C:/pywps_processes'

importwps
```

Step 5:

Install the PyWPS Python package, open the Windows command line tool (start > run), drag and drop 'setup.py' from your 'c:\pywps' folder, type install after the location of the setup file and click ok. The PyWPS Python package is now installed.

Step 6:

[7] This tutorial is also published on the Deltares Wiki: https://publicwiki.deltares.nl/display/OET/Setting+up+PyWPS+in+a+Windows+environment

Test your WPS by starting Apache server and navigate to http://localhost/cgi-bin/pywps.cgi
The result should be something similar to this:

Step 7:

Add your own Python processes as WPS by copying your processes to the 'c:\pythonprocesses' folder.

```
<ExceptionReport xmlns="http://www.opengis.net/ows/1.1" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-
instance" version="1.0.0" xsi:schemaLocation="http://www.opengis.n
et/ows/1.1
http://schemas.opengis.net/ows/1.1.0/owsExceptionReport.xsd">

    <Exception exceptionCode="NoApplicableCode">

        <ExceptionText>'No query string found.'</ExceptionText>

    </Exception>

</ExceptionReport>
```

In 'c:\pywps\tests\processes' some example processes are included, copy these to your 'c:\pythonprocesses' folder and find out whether you can get them to run.

Check whether all processes are recognized by doing a GetCapabilities request (http://localhost/cgi-bin/pywps_test.cgi?service=wps&request=GetCapabilities).

Note: PyWPS can also be used with the Q-GIS WPS plugin byDr. Horst Duester

Official documentation on PyWPS (v3.2.0) can be found here:
http://pywps.wald.intevation.org/documentation/pywps-3.2/

II. Implementation of WPS Processes

The chosen WPS server instance for this project is PyWPS. The implementation of processes as WPS is made in a Python function; these scripts have a quite uniform syntax. All scripts in PyWPS require an input, process and output section. This chapter describe the basic layout of these sections and also the layout of the produced scripts.

Script build up

The processes to be disseminated via PyWPS are built like regular Python scripts although some additional information should be included. This information is mainly meta-data for identification and capabilities of the process and definition of in- and outputs. The following sections describe how scripts are written.

The script used as WPS process should first import the required PyWPS module. Importing the module informs Python about the functions required in the script. The code required for doing this can be found in Textbox 9.1.

```
from pywps.Process import WPSProcess
```

Textbox 9.1 - Importing of the required PyWPS functions.

The next step is defining the process' meta-data. This data is used in the 'DescribeProcess' request output and can contain an extensive description of the process. In this section the identifier, title and version are given. Also the process' version can be defined here, as well as the abstract, support for storage and status supply are set. The identifier is probably the most important since that is the parameter which is used to call the process, for a correct identification of the process; the identifier should be the same as the name of the Python file. An example of the meta-data code can be found in Textbox 9.2.

```
class Process(WPSProcess):
    def __init__(self):
          WPSProcess.__init__(self,
          identifier = "ProcessID",
          title = "Process Title",
          version = "Version#",
          storeSupported = "true",
          statusSupported = "true",
          abstract="Description of the process")
```

Textbox 9.2 - Code used for definition of the process' metadata.

Defining Process' Inputs and Outputs

For use in processes PyWPS allows three types of in- and outputs. These types represent the way data is expressed. Allowed data types are complex, literal and bounding box, which are represented by the tags 'addComplex', 'addLiteral' and 'addBBox', an example of how these data types are defined in the script is given in Textbox 9.3.

```
self.input1  = self.addComplexInput(identifier = "input1ID",
                                title = "Input Title")
self.input2  = self.addLiteralInput(identifier = "input2ID",
                                title = "Input 2 Title",
self.input3  = self.addBBoxInput(identifier = "input3ID",
                                title = "Input 3 Title")
self.output1 = self.addComplexOutput(identifier = "Output1",
                                title = "Output 1 Title",
self.output2 = self.addLiteralOutput(identifier = "Output2",
                                title = "Output 2 Title")
self.output3 = self.addBBoxOutput(identifier = "Output2",
                                title = "Output 3 Title")
```

Textbox 9.3 - Code used for definition of the process's in- and outputs.

The naming of the data types is quite straight forward. 'Complex' type refers to an input file. This can be any file type that can be implemented by the script. Since the WPS standard is relatively general there are no restrictions to the file types. The 'Literal' type refers to any kind of literal data such as floating numbers, integers or strings. As input these variables are given by the user and as output the variables are printed in the result.

The most 'geographical specific' data type is the 'BBox' data type. This type refers to a bounding box represented by two sets of coordinates, the first for the lower left corner and the second for the upper right corner of the bounding box. The bounding box can for input be used for example to specify the application area of the process. As an output the bounding box can for example indicate which area is affected by a certain process.

Process

Scripts being implemented as WPS may already be available on beforehand and offer certain functionality on a local PC or in a different kind of processing service. In the case this script is a Python script, the implementation is very straightforward. In the basis build up of the WPS script a part is defined where the actual script is being implemented. The line that announces part of the script is given in Textbox 9.4.

```
def execute(self):
    # Script
```

Textbox 9.4 - Code used for defining the actual process code, execute code. The '# Script' part should be replaced by the actual code of the process to be implemented.

Instead of the ‘# Script’ part the actual code is placed. There are a few things that should be taken into account when adding your code to a general WPS script are the following. The added code is part of ‘execute(self)’ so the code has to be indented. A second reminder is to change the parts of the code where inputs are required, or add the input variables to the variables used in your code, see also Textbox 9.5.

```
a = self.Input1.getValue()
b = self.Input2.getValue()
```

Textbox 9.5 - Example of coupling the WPS input values to the input values originally used in the code.

Process Result

In the process’ output XML schema the process result is given. The process output is already defined in 0 but the output has to be linked to the actual output of the script. In the example in Textbox 9.6 it is shown how this is done. In the example the ‘out’ variable is the variable where the result of the script is stored in. This variable is then used as the value to be given to the WPS’ output variable.

```
self.output.setValue(out)
return
```

Textbox 9.6 - Code used for setting the process's output. In the example the variable ‘out’ is defined to be the result from the script, this result is now coupled to the process output parameter.

A complete example script is in Textbox 9.7 This process squares the input value and gives the result as output. In the script three areas are identified. The green square indicates metadata of the process, in the blue box in- and output parameters are indicated and in the red box the actual operation is shown. The URL for the Execute request of the example script is in Textbox 9.8.

```
from pywps.Process import WPSProcess
from types import FloatType

class Process(WPSProcess):
     def __init__(self):

        WPSProcess.__init__(self,
             identifier       = "input_squared",
             title            = "This process squares the input value",
             version          = "1",
             storeSupported   = "true",
             statusSupported  = "true",
             abstract         = "test process")

        self.input  = self.addLiteralInput(identifier  = "input",
                                           title       = "Input Value",
                                           type        = FloatType)
        self.output = self.addLiteralOutput(identifier = "output",
                                           title       = "Output Value",
                                           type        = FloatType)

     def execute(self):
        val = self.input.getValue() * self.input.getValue()
        self.output.setValue(val)
        return
```

Textbox 9.7 - Example script of a PyWPS process that squares the input value. The green box contains metadata of the service and server based parameters; the blue box contains the definitions of in- and outputs; the red box contains the actual operation.

```
http://localhost:8080/wps.cgi?
     request=Execute&
     service=wps&
     identifier=input_squared&
     version=1.0.0&
     DataInputs=[
            input=8
            ]
```

Textbox 9.8 - URL to execute the example script. Input value is set to 8, the expected output is 64.