Evaluation of the OGC Web Processing Service for Use in a Client-Side GIS

Christopher Michaelis[1] and Daniel P. Ames, PhD, PE[2]

Abstract

The Open Geospatial Consortium Web Processing Service proposed specification is intended as a solution for developing web-based geoprocessing plug-ins, and for easily sharing algorithms and geoprocessing functionality. This paper seeks to evaluate the WPS proposal with respect to feasibility and potential utility, and to identify areas for improvement. Challenges with the WPS proposal are discussed together with potential solutions. Several potential enhancements to the WPS proposal are introduced and considered, including a mechanism to guide client applications in prompting for correct data and a means to list the data available on a server.

1. Introduction

The Open Geospatial Consortium (OGC, or OpenGIS) is a consensus standards organization concerned primarily with the release of open (i.e., non-proprietary) specifications to unite geographic information software, bringing the multitude of disjoint formats and communications mechanisms together to allow interoperability (Open Geospatial Consortium, 2006). Rather than avoiding this standardization and remaining fully proprietary, many GIS system developers "have shown extraordinary cooperation in teaming to submit OpenGIS Specifications" (Information Today, 1997) and have actively embraced the standards, some even participating in their development. As the OGC is composed of many professionals in multiple fields, rather than a single committee in a single corporate environment, the standards are typically of consistently high quality and are suitable for any number of differing scientific tasks.

On November 17, 2005, the OGC released the fourth revision of a proposal for a specification called the Web Processing Service (WPS) (Open Geospatial Consortium, 2005). This proposed specification describes a mechanism by which geoprocessing may be performed on remote servers, using principally extensible markup language (XML) for communication through the Internet. The specification is authored in such a way that it should be fully language and platform independent. The OGC requested public comments for a time, with a cutoff date of February 4, 2006. Although the forum for public comments has already closed, there have been, to date, few if any real-world studies on the feasibility and utility of the proposal from the client-side GIS point of view.

Prior to WPS, web-based geoprocessing systems and approaches similar to WPS have been implemented by various entities. Most notably, the Environmental Systems Research Institute (ESRI) product ArcInfo 8.3 (ESRI, 2003) contains a feature called the Geoprocessing Server, which ran on large-scale UNIX servers to perform geoprocessing on behalf of ESRI client software which submitted jobs for processing. The ESRI Geoprocessing Server protocol is proprietary and closed such that only ESRI software is able to make use of the remote processing capabilities. Interestingly, this feature was removed from the following version (ArcGIS 9.0). A similar but subtly different feature was introduced in ArcGIS Server 9.2, where a "Model Builder" tool constructed from simpler ESRI geoprocessing components may be served to ArcGIS Desktop and ArcExplorer clients (Environmental Systems Research Institute, 2006). Unlike WPS, the ESRI implementation is not compatible with non-ESRI products and a closed, proprietary communications protocol preventing it from being adopted at large or studied in a non-ESRI environment.

2. What is the Web Processing Service (WPS)?

The Web Processing Service defines a mechanism by which a client may submit a processing task to a server to be completed. The service defines a "server instance", or server, as an entity which may provide one or more processes, or individual processing tasks (e.g., adding two raster datasets together could be one process). In this manner, any given server may be able to perform multiple different, and not necessarily related, processes.

The specification indicates that extensible markup language (XML) should be used for all communication. Extensible markup language documents are made up of individual elements, which are logical containers for

[1] Research Assistant, Idaho State University Geospatial Software Lab
[2] Assistant Professor, Dept. of Geosciences, Idaho State University

related data. An element may contain other elements, and any given element may contain attributes which describe that element. A simple example of an XML document might be:

<landscape name="Smithsonian Park">
<tree type="Elm" height="8" />
</landscape>

This XML document describes a landscape, which is indicated by the element entitled "landscape". An attribute entitled "name" indicates that this is the Smithsonian Park. A sub-element entitled "Tree" implies that the landscape contains or owns this tree, and the tree has a further two attributes which describe the type of tree and its height. The element is closed, or ended, by repeating the name of the element with a leading slash.

XML is designed to be "straightforwardly usable over the Internet", "human-legible and reasonably clear", "formal and concise", and "easy to create" (W3C, 2000). Using XML is beneficial mainly due to being human-readable, which assists greatly in designing and debugging applications using it. XML documents may be validated to ensure that they contain all needed elements and attributes. Validation takes place against an XML schema, which is a specialized form of XML document which describes the structure that an XML document must follow.

The main goal of the Web Processing Service is to define how to communicate to perform remote processing. To this end, there are three key requests which may be made of a WPS server: GetCapabilities, DescribeProcess, and Execute. The first of the requests asks the server to list the individual processes which are available on that server, along with a short abstract and keywords. The request does not require any parameters. Once a process has been identified from the response, a "DescribeProcess" request may be sent, providing the process in question as a parameter. The response to this request includes the same information as the GetCapabilities response, plus more detailed information about any needed input parameters for the process and whether the input is simple (e.g., a simple number like 23) or complex (e.g., a data file). Complex outputs are typically encoded as XML, for instance using GML (Geographic Markup Language, a relative of XML) for vector data.

If the DescribeProcess response indicated that this is the process the user or client wishes to execute, the third request (Execute) may be invoked. This requests that the server actually performs the operation. Necessary parameters include the name of the process as well as any applicable inputs for the particular process. The response to the Execute request is an ExecuteResponse document, another XML document which indicates a process status, indicates the inputs that were used, and provides either simple literal value outputs or links to complex outputs. The process status may be "ProcessAccepted", indicating that the process was received and is in queue to be processed; "ProcessStarted", indicating that the process is underway; "ProcessSucceeded", meaning the process completed; or "ProcessFailed", indicating that a problem occurred. If the status is ProcessAccepted or ProcessStarted, the status is accompanied by an attribute which indicates where the next ExecuteResponse document may be found. In this way, the client may check on the status of the process by requesting the next ExecuteResponse document. In the case of ProcessStarted, a status message and progress percentage may also be provided.

If the process status is ProcessFailed, the ExecuteResponse document also contains an error code embedded in an XML ExceptionReport element, which may be one of five error codes (MissingParameterValue, InvalidParameterValue, NoApplicableCode, ServerBusy or FileSizeExceeded). If the process succeeded, the response document will also include either the outputs (in the case of simple literal values) or URL links to complex outputs (such as a file with raster data). If a single complex output is produced, that output may be returned directly in lieu of an ExecuteResponse document. Together, these three operation requests and their responses constitute the majority of the Web Processing Service proposal.

3. Why and when should a Web Processing Service be used?

Because the geoprocessing functionality proposed is unlimited in scope or nature (Open Geospatial Consortium, 2005), the proposal holds great promise for utilizing computational tools without traditional concerns such as distributing bug fixes or checking for the most up-to-date version. Although the scope of what may be accomplished is unlimited, many operations can be completed more quickly locally (i.e., on a user's

desktop PC) than remotely (i.e., on a central server), especially after factoring in time to upload input data and subsequently download resulting outputs. When determining whether to use local or remote data processing, a number of factors must be considered beyond the raw size of the datasets involved. Computational complexity plays a large part. If the process takes several hours to complete even on a small dataset, it can be better to process the data remotely. If the task is not complex and the bulk of the work lies in pressing through large volumes of locally stored data, it can be more efficient to perform the processing locally. Hence, before one chooses to use only remote data processing, an intelligent choice ought to be made whether to proceed with processing locally or remotely. As illustrated in the simple graph in Figure 1, remote processing has a useful place in modern computing. Higher processing power in server farms can easily reduce the cost of time-demanding and highly complex tasks, especially when combined with high-throughput networks such as fiber channel or gigabit ethernet. Therefore, data should be sent to servers typically having higher processing power, instead of using the slower local computer, when the time to process the data locally would be greater than the combined time to transmit the data, process it remotely, and download it again.

Remote processing is also an ideal choice if the algorithm is relatively new, and is under active development. In this scenario, new version releases do not require software upgrades by end users. Rather, only the server requires an update to reflect the new code or algorithm. Subsequently all users automatically gain access to the most current and most accurate version of the process simply by using the server-based processing service. This single point of control over the process also creates the possibility of charging for the use of the process, if desired.

The traditional view of remote processing mandates that input data is uploaded, as in Remote Procedure Calls (RPC) where input data and parameters are sent together with a function call (Bloomer, 1992). Input data could be stored on the server as well, requiring the client only to specify the particular input which is desired. This creates the opportunity for a processing server to also provide data: raw data wouldn't necessarily be provided, but processed result data could be returned without needing to upload input data. This makes a great deal of sense particularly for processes requiring real-time data such as weather station observations, live traffic observations, etc. These processes could be provided by the same agency that collects the data, allowing the processes to have access to the latest available data at all times. The motivations for using a remote processing server are many, but ultimately the decision must lie with the user whether remote processing is appropriate for the task.

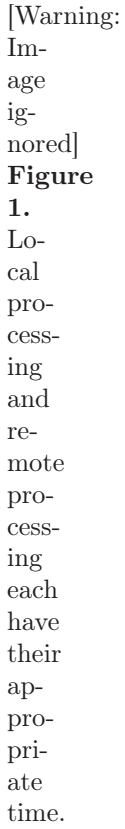

Figure 1. Local processing and remote processing each have their appropriate time.

4. WPS Implementation Considerations

The WPS proposal describes a mechanism by which a client computer may submit a job to be processed on a server computer. This is classic client/server architecture, meaning that both a client component and a server component are needed. For implementation and testing purposes it is useful to build the client-side component on top of an existing geographic information system to take advantage of existing visualization features for the geospatial data to be processed and returned, however, this is not always required and initial testing may easily be performed with command-line or spatially unaware testing tools. This client-side component is the portion which handles XML communication through the internet with the server, ideally without the user needing to directly see or work with the XML at all in order to discover available processes or to make their request and retrieve results. It is desirable to build this client-side component in such a way that it is not tied to any specific software package, proprietary or otherwise.

The server-side component which provides the client implementation (and other client implementations) with actual services should similarly be standalone (i.e., not tied to any particular geoprocessing algorithm or process that is to be provided). One approach to this problem is to place an interface layer around existing or new geoprocessing routines written as command-line Linux-based utilities, Windows services, or Windows applications. As long as the application or algorithm implementation in question may be executed without user interaction (e.g., through command-line arguments, TCP/IP communication, or through OLE data transfer), this thin communications layer is a good option. It may be placed around existing geoprocessing functionality and existing tools (or newly developed tools) to enable them to be served using WPS, by providing XML that meets the interface requirements of the WPS communication schema. Rather than requiring a full rewrite of existing software, the existing software will likely only need minor modification to make it able to run in an unattended mode, ideally via command-line arguments and generated log files. GRASS utilities (Neteler, 2006) are an excellent example of existing command-line tools which may be wrapped using an approach such as this.

This wrapper may then initiate the tool as needed, and monitor a log file or a return a status message to indicate whether the process succeeded, is still running, failed or is in queue. Status percentages and relevant

error messages may also be retrieved from log files, to be returned to the end user through appropriate communications in an automated fashion, with this wrapping technique. Placing this thin wrapper around standalone tools enables them to be used in multiple places and for multiple purposes with a minimum of effort. In essence, this wrapper becomes the "WPS Server", because it handles all communications needed by WPS. This wrapper, constituting the WPS server, could be implemented as a PHP web page, as an ASP.NET web page, as a standalone application or using any other server technology.

Many of the operations needed by a WPS server are simply metadata operations: providing information about individual processes (i.e., required inputs) and listing processes available on a server. The WPS server will ideally load information on available processes from a configuration file (perhaps one per process) or from a database, thus making the code written for the WPS server reusable simply by adding additional processes to configuration files or to the database. These configuration files or this database may also indicate to the WPS server how to launch the process and how to parse its output files.

An overview of the suggested communications flow between a client application and a server application is shown in Figure 2. Initially, the client application will query the server, providing a GetCapabilities request. The WPS server then returns an XML document matching the WPS schema (arrow 1 in Fig. 2). The client then presents the user with a list of the processes available on the server (arrow 2 in Fig. 2). The user then selects one of these processes, causing the client to request additional information on that process (arrow 3) by sending the DescribeProcess request. The client application helps the user to collect and enter any needed input parameters for the process to be executed; it then initiates the process on the server (arrow 3) by sending the Execute request. The server in turn will trigger a specialized process launcher utility (arrow 4).

Note that the WPS server should not execute the requested process directly, because this would imply a delay: the server would need to complete the process before a reply to the client could be sent. Instead, it is desirable to have the WPS server initiate processing in a different thread, or in a different process, so that an initial ExecuteResponse may be immediately returned to the client. In this manner, geoprocessing operations which are expected to take a good deal of time will still allow the end user to use their computer. For cleanliness and safety, it is best to start a completely new process, so that if a process terminates abnormally it won't stop the entire WPS server as well: hence, a specialized process launcher utility is a good solution.

[Warning: Image ignored] **Figure 2.** Suggested communications flow between the client and the server.

As explained in section 2, this initial ExecuteResponse document informs the client application of where future status updates (i.e., the next ExecuteResponse) on the requested operation may be found, as well as notifying of success, failure, accepted or rejected jobs, and providing status updates. The client application should request this next ExecuteResponse document as often as the user wishes (arrow 6 in Fig. 2). Ideally the WPS server should be designed such that an ExecuteResponse document may be found in a consistent location, perhaps always using the same URL for the most recent response. A good approach to addressing generation of response documents would be the creation of a specialized web page or server component which simply parses a log file being written by an executing process to determine current status or errors.

Having the ability to provide status updates requires that the server component is able to get such updates from the process which is executing. As implied, a log file is an excellent solution to this: the process should be able to write a log file or to redirect its standard output to a text file. In this manner, the file may be automatically parsed by the WPS server, providing good connectivity between the components of the system.

Once the ExecuteResponse document indicates that the process has completed, the server must return any applicable error message or return information on where the generated data may be downloaded or obtained. The client software should then allow the user to save or view the data, thus completing the role of WPS.

5. Suggested Enhancements, Problems and Potential Solutions

While the WPS proposal is able to accomplish its stated goals in its current form, the proposal could be enhanced by six key changes. These changes include two additional elements in the DescribeProcess response provided by a server, which describes a given process's inputs and outputs, as well as a mechanism by which a client may cancel a request which is pending or processing. Potential changes also include correcting some inconsistencies in behavior, providing additional exception types for error handling, and having only a single entry point for each process and perhaps a single entry point for each server.

The first suggested change is to add an element to the Extensible Markup Language (XML) document which is returned by a DescribeProcess request. Currently, the needed inputs are listed by this document, but no clear description of how the client should prompt the user for this input is provided. The needed data may come in the form of selecting a shape on a map, providing a literal value such as "23", browsing for a file of a given type, or dozens of other methods for collecting data. In our test implementation, we introduce a new XML element called "PromptMethod" to address this problem. The element may contain the values "browseforvector", "browseforraster", "getboundingbox", or "getmatchingregex". These will cause the client application to prompt for a vector file, prompt for a raster file, retrieve a bounding box (e.g., by asking the user to draw it) or collect a piece of information matching a particular pattern, respectively. The first three require no explanation; the last, getmatchingregex, will accept only a user-entered value which matches the provided regular expression.

A regular expression is a rule defined by special characters, such as "^[a-zA-Z][a-zA-Z]$". This regular expression would look for the beginning of the input (symbolized by ^), followed by two letters, from A to Z, independent of case, followed by the end of the input (symbolized by a dollar sign). This provides a flexible and quickly configurable input filter to ensure that a user enters only input that is suitable. Some variations for syntax in regular expressions exist due to differing regular expression processing engines (Goyvaerts, 2006), meaning that care should be taken to ensure expressions are designed in such a way that they may be interpreted in the same way on both clients and the server. An example of the suggested addition to WPS may be seen in Figure 3, where the regular expression is defined as "^\d{8}$". This expression indicates that the start of the string (^) should be followed by eight digits (d{8}) and the end of the string ($). This expression assumes that Microsoft's regular expression processor will be utilized.

The second suggested change is another addition to the DescribeProcess response. Currently there is no mechanism by which a server may list the data that is available on the server for use by a given process. An excellent example of where this may be useful is in the processes of watershed delineation - defining stream networks and watersheds based on raster elevation datasets (Savant et al., 2002). Elevation raster datasets can typically be very large, so it is undesirable and often impossible to upload the entire input file to the server. Datasets such as elevation data typically do not change often, and may be stored on the server to save

time. In our test implantation, we address this by introducing an XML element entitled “AvailableData”, with a child element for each data item containing a name and a brief description as shown in Figure 4. Not only does this speed up processing by removing the need to upload the input data, but it also creates a means by which a WPS server may act as a data repository as well as processing it and returning that processed data.

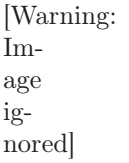

Figure 3. Suggested Prompt-Method element in DescribeProcess response.

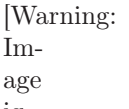

Figure 4. Suggested AvailableData element in DescribeProcess response.

In the current WPS proposal there is an inconsistency after submitting a job to a server with regards to what should occur next.. If the process will result in a single return value and the Execute request was made with the "store" parameter set to false, WPS will allow the process to immediately return the output, rather than an XML ExecuteResponse document. If there is more than one output, or if the process has been asked to store the results, then an ExecuteResponse document is generated. This behavior is inconsistent, since any given process might return either multiple outputs or a single output, depending on the parameters provided to the process. Instead, it is simpler and more straightforward to always return an ExecuteResponse document, storing any complex output data (e.g., vector or raster files) on the server until downloaded by the client. Simple value outputs (e.g., a single number) may be returned directly embedded in the ExecuteResponse document, with links pointing to complex outputs. Our third suggested change is to require always returning an ExecuteResponse document to provide greater consistency and simplify client implementation far easier.

After submitting a job to a WPS server, it may be desirable to cancel the requested operation; this eventuality is not planned for with the current WPS proposal. In our test implementation (and as our fourth suggested change) we introduce the use of a "cancel request URL" in the ExecuteResponse document, along with the existing URL indicating where the next ExecuteResponse document may be found. In this manner a client wishing to cancel a process needs only to access the URL to trigger a cancellation of the requested process, preventing wasted server processing time on undesired processing.

Our fifth suggested change to the WPS proposal is to have a single entry point for each possible request (GetCapabilities, DescribeProcess, and Execute) on a given process. Ideally, a single entry point (e.g., a single PHP web page) could be used not only for every request on a process, but also for every process available on that server. Presently, each of the requests that a process supports may have a different URL to perform that request, as illustrated in Figure 5. In the figure, the GetCapabilities URL is circled in blue, the DescribeProcess URL in green, and the Execute URL in red. Here each URL is the same, making maintenance and implementation easier, although it is acceptable to have a different access URL for each operation. Confusion or errors can easily arise with differing URLs for these operations. This can be addressed easily by making it required to use only one URL for all three of the requests that a process must support.

Lastly, our sixth suggested change is to implement a more highly structured exception system. Presently there are only a few exception types (MissingParameterValue, InvalidParameterValue, NoApplicableCode, ServerBusy and FileSizeExceeded), which are limiting - particularly when attempting to parse the error automatically. With a small number of exception types (exception codes), there is no generic way in which to handle errors on behalf of the user; the only thing that can be done presently is to display the error to the user. A more complex and structured exception hierarchy would allow client applications to understand the nature of the error that occurred, perhaps recovering or retrying automatically as needed. The ability to insulate the end user from errors and recover gracefully is an important part of any standard design, and this ability would be made possible with a more detailed exception hierarchy.

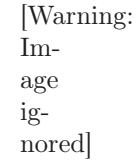

Figure 5. Ability to have a differing URL for each request on the same process can cause confusion and unnecessary complexity.

6. Conclusions

Overall, the WPS proposal was found to be an effective way of standardizing on a communications mechanism for client/server geoprocessing. The proposal works as currently designed, and is indeed suitable for many GIS tasks. Implementation of the client component and the server-based WPS processes showed that the WPS proposal is sound and effective. Nonetheless, there are several opportunities for enhancement to the WPS proposal including the six specific recommendations provided here. It is hoped that these observations will benefit others working to implement the WPS proposed specification on various GIS platforms.

Any conceivable geoprocessing activity may be presented in a WPS format. This allows the developers of the algorithm to continuously improve the algorithm while still allowing users at any location to use the code, without needing to update to a latest version. This also ensures that code and algorithms being used are of high quality; if a bug is detected in a released version of an algorithm which is distributed on CD,

no easy way exists to ensure that all users successfully upgrade to the corrected version. With a web-based processing architecture, this problem is alleviated by simply publishing the new algorithm to the processing server. Processor intensive and time consuming geoprocessing may be performed on remote servers, freeing the user's desktop to work on other tasks. Large datasets may even be stored on the server and processed according to the user's particular parameters, combining data repository and data processing aspects into one system.

The WPS proposal introduces a standard which will allow diverse developers at different locations to produce geoprocessing offerings and provide them easily to differing client platforms. In these tests, the WPS proposal has been shown to be practical and usable in its current design. The additional enhancements proposed can improve WPS significantly, making it better suited still to the task for which it was designed.

7. References

Bloomer, John. 1992. Power Programming with RPC. Cambridge, MA: O'Reilly Media.

Environmental Systems Research Institute (ESRI). 2003. ArcGIS Desktop Products Data Sheet. WWW document, http://www.esrichina-bj.cn/produce/esri/arcgisdesktopsheet.pdf

Environmental Systems Research Institute (ESRI). (2006). ArcGIS 9.2 Webinar – ArcGIS Server: Publishing a Geoprocessing Model. WWW document, http://events.esri.com/info/index.cfm?fuseaction=seminarRegForm&show

GIS Competitors Cooperate on OpenGIS Specs. 1997. Information Today, 14(2), 15-15.

Goyvaerts, Jan. 2006. Regular Expression Tutorial. WWW document, http://www.regular-expressions.info/tutorial.html

Open Geospatial Consortium, Inc. 2006. Vision and Mission. WWW document, http://www.opengeospatial.org/about/?p

Open Geospatial Consortium, Inc. 2005. OpenGIS® Web Processing Service (WPS) Discussion Paper. WWW document, http://www.opengeospatial.org/