

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

Christopher D. Michaelis · Daniel P. Ames

Received: 26 March 2007 / Revised: 1 February 2008 /
Accepted: 25 February 2008 / Published online: 1 April 2008
© Springer Science + Business Media, LLC 2008

Abstract The Open Geospatial Consortium Web Processing Service specification is intended as a solution for creating and distributing web-based functions. This paper seeks to evaluate the WPS specification with respect to feasibility and utility, to identify areas for improvement, and to provide a demonstration implementation approach including a client-side tool and a server-side wrapping technique for exposing geoprocessing functionality through WPS using an asynchronous approach. Challenges with the WPS specification, some of which are already being addressed in the newest WPS revisions, are discussed together with proposed 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.

Keywords OGC · WPS · geoprocessing · geographic information systems

1 Introduction

The Open Geospatial Consortium (OGC) is a consensus standards organization concerned primarily with the release of non-proprietary specifications to unite geographic information software, bringing together a multitude of disjointed formats and communications mechanisms to allow interoperability [8]. Rather than avoiding the OGC standardization process and remaining fully proprietary, many geospatial technology developers “have shown extraordinary cooperation in teaming to submit OpenGIS Specifications” [4] and have actively embraced the standards, many participating in their development. As OGC is composed of professionals in multiple fields, rather than a single committee in a single corporate environment, the standards typically are of consistently high quality and are suitable for any number of application areas.

C. D. Michaelis (✉) · D. P. Ames
Department of Geosciences, Idaho State University, 199 Valleyview Drive, 921 South 8th Ave.,
Pocatello, ID 83209-8072, USA
e-mail: cmichaelis@happysquirrel.com

On June 8, 2007, version 1.0.0 was approved to become an official OGC standard called the Web Processing Service (WPS) [10]. This standard specification describes a mechanism by which geoprocessing may be performed on remote servers, principally using 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. During initial design, OGC requested public comments, with a cutoff date of February 4, 2006. Although the forum for public comments has already closed, there have been, to date, few real-world studies on the feasibility and utility of the specification from the client-side GIS point of view. We seek to provide feedback on the specification, with tools based on the state of WPS during the proposal period.

Prior to WPS, web-based geoprocessing systems and approaches similar to WPS were implemented by various entities. Notably, the Environmental Systems Research Institute (ESRI) product ArcInfo 8.3 [2] contains a feature called the Geoprocessing Server, which used 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 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 “ModelBuilder” tool constructed from simpler ESRI geoprocessing components may be served to ArcGIS Desktop [2] and ArcExplorer [3] clients. This was a substantial contribution to the field, and added significantly to further developments in this area. 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. In addition W3C service solutions, Java servlets, and similar web processing services that are not specifically GIS oriented have contributed both knowledge and strategy to web based geoprocessing attempts.

2 WPS overview

The Web Processing Service specification 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 XML may be used for almost all communication, with the notable exception that WPS version 0.4.0 does not allow XML to be used in the GetCapabilities request. XML documents are made up of individual elements, which are logical containers for related data. An element may contain other elements, and any given element may contain attributes which describe that element.

XML is designed to be “straightforwardly usable over the Internet,” “human-legible and reasonably clear,” “formal and concise,” and “easy to create” [14]. XML is beneficial due to its human-readability, 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, a specialized XML document for validating other XML documents.

The main goal of the Web Processing Service is to define an XML based communication protocol for remote geoprocessing. 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 selected, a DescribeProcess request may be sent. 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., an integer) or complex (e.g., a data file). Complex outputs are typically encoded as XML, using Geographic Markup Language (GML), an XML grammar, for encoding and communicating geospatial content for vector data.

The third request (Execute) may be invoked, requesting the server to perform the selected operation. Necessary parameters for the Execute request 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 process status, inputs that were used, and 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 WPS protocol.

3 Application of WPS

Because the potential geoprocessing functionality supported by the WPS protocol is unlimited in scope or nature [10], WPS holds great promise for using computational tools without traditional concerns such as distributing bug fixes or checking for the most up-to-date code version. However, some geoprocessing operations can be completed more effectively 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 (or transfer it from another server) 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, an informed decision needs to be made regarding whether to proceed with processing locally or remotely for any particular task. 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 fibre channel or gigabit ethernet. Therefore, data should be sent to servers typically having higher processing power, instead of using a 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. Some of this time delay may be offset by allowing a request to a WPS server to specify an alternate data source rather than a direct upload, (e.g., a Web Feature Service GetFeature request). In this case, the WPS would download the required data from another server rather than receiving it from the WPS client. Such a process could become common in service-oriented architectures.

Remote processing is also appropriate for deployment of new algorithms and code that 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 generating revenue from the process.

The traditional view of remote processing requires that input data is uploaded, as in Remote Procedure Calls (RPC) where input data and parameters are sent together with a function call [1]. However, input data could also be stored on the server, requiring the client only to specify the particular input which is desired. This creates the opportunity for a WPS server to also serve raw or pre-processed data. This approach could be particularly useful for processes requiring real-time data such as weather station observations or live traffic observations. WPS services could be provided by the same entity that collects the data, allowing the processes to have access to the latest available data at all times. For example, a WPS could retrieve vehicle location data from a server, re-project it to a new coordinate system, and return the result. Another WPS could perform advanced terrain analysis on server side terrain data returning only the few desired derivative results. 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.

4 WPS implementation considerations

The WPS protocol describes a mechanism by which a client computer may submit a job to be processed on a server computer, using uploaded data or data provided via a WFS or WCS service. 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 a geographic information system (GIS) to take advantage of existing visualization features. However, initial testing may be performed with command-line or spatially unaware tools. The 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 to discover available processes or to make their request and retrieve results.

One approach to generic development of the client side component is to place a reusable interface layer or a “wrapper” around existing or new geoprocessing routines written as command-line Linux-based utilities, Windows services, or Windows applications. If the application or algorithm implementation in question may be executed without user interaction (e.g., through command-line arguments, network communication, or through inter-application data transfer), then this thin communications layer is a useful option.

Wrappers 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. Existing software will likely only need minor modification to enable it to run in an unattended mode, ideally via command-line arguments and log files. GRASS utilities [7] are an example of existing command-line tools which may be wrapped using an approach such as this.

The server-side wrapper may 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, and returned to the end user through appropriate communications, with this wrapping technique. Placing a thin wrapper around standalone tools enables them to be used in multiple places and for multiple purposes with minimum effort. In essence, the wrapper becomes the “WPS Server”, because it handles all communications needed by WPS. The WPS server could be implemented as a PHP web page, as an ASP.NET web page, as a standalone application, or implemented using any other server technology.

Many of the operations needed by a WPS server are essentially 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 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.

In the following section, we describe a test implementation of the WPS protocol, including a strategy for implementing services based on the considerations described above. An alternative but similar approach followed by the 52°North Initiative [15] was used to produce a Java-based implementation with support for all WPS 0.4.0 features.

5 test WPS implementation

With a plug-in architecture supporting the Visual Basic.NET and C# programming languages (our preferred programming languages), MapWindow GIS [6] was a useful environment in which to develop a test implementation of the WPS protocol. Specifically, we constructed a MapWindow plug-in called the “WPS Gateway” that allows the end user to browse a list of available WPS servers from a simple database-driven catalog (used to store a listing of available WPS service URLs), as well as the processes available on those servers. Any given remote process is referred to in this system as a “WPS Online Plug-in”, which is accessed via the WPS Gateway component. The server-side interface layer (process wrapper) described in the design is referred to as a “WPS Online Plug-in Wrapper” in this implementation. For the purposes of this test, two unique wrappers were constructed, one for use with Linux using PHP and one for use with Windows using ASP. Net.

The Linux implementation involved creating a PHP page to provide metadata information in response to GetCapabilities and DescribeProcess requests and a PHP page to provide response documents. Additionally a C++ process launcher application was created. Each of these tools may be reused for other processes as well, because the same PHP files may be reused for multiple processes. Adding a new process can be done by adding a new section to each of the two PHP files which is customized to the new

process. This approach can be extended so that the utilities do not need to be modified for new processes, but instead a configuration file is saved describing each process provided by the server. The configuration files will then be read by the tools, making direct editing of the source code files for each new process unnecessary. This has the benefit of keeping WPS functionality centralized and abstracted from the geoprocessing tools, as well as providing the convenience of having a single entry point for all WPS processes on a given server. Taken together, these utilities constitute a “WPS Online Plug-in Wrapper” for Linux.

To access the WPS server from MapWindow GIS, the user enables the WPS Gateway plug-in from a menu. This causes several menu items to be added to MapWindow including a “Browse Catalog” menu. Because there are currently no central catalog services available online for listing available WPS processes, a simple database-driven catalog service has been developed for use by this tool.

Upon choosing “Browse Catalog”, the gateway plug-in queries the catalog service on a central server to fill a list of available servers. When the user selects one of these servers, the gateway queries that server with a GetCapabilities request, which returns the server capabilities listing the available processes. The gateway fills a list with these processes. The user may then select one of these processes, causing the gateway plug-in to request additional information on that process via a DescribeProcess request. If the user decides to execute this process, the gateway plug-in first collects any needed input data or parameters from the user. It then initiates the process on the server by sending an Execute request. The server in turn triggers the specialized process launcher utility as designed. This process launcher uses the “fork” operation to initiate the new process which is not connected to the web page. The initially started process is then free to continue processing while the server completes successfully to return an initial ExecuteResponse document to the gateway plug-in, after which the user is informed that the process has been accepted by the server. The user may continue using their computer while execution continues in the background on the server past this point.

The newly created server side process resulting from the fork operation then proceeds to launch the actual operation, which executes and writes a log file in the process. Periodically, the gateway application will check on progress. This status check operation triggers another web page, called “GetStatusUpdate”, which parses the contents of the log file being written for the particular process request, returning an appropriate status message and progress percentage. This occurs numerous times until the status indicates that the process has completed or failed; in the case of failure, the error message is displayed to the user and the process halts. If successful, the gateway plug-in downloads any output data and causes it to be displayed in MapWindow GIS.

Two WPS processes were built for this test client implementation to consume. The first was written for Linux using C++, and performs basic raster mathematic operations (addition, subtraction, multiplication and division). A process such as this may benefit greatly from the WCS (Web Coverage Service) draft standard [9], being aligned with the WPS standard, as it focuses on transfer of raster data.

The second online plug-in used to demonstrate and test the WPS proposal is a watershed delineation tool. This tool uses Terrain Analysis using Digital Elevation Models (TauDEM) [13] to identify drainage area in terrain data. The process accepts a raster dataset with elevations as input and produces several intermediate outputs as well as a stream network shape collection and a watershed polygon shape collection representing flow patterns and drainage area boundaries for the area. Implementation of the watershed delineation online

plug-in wrapper involved creating a new Windows service called “Delineation Listener” which monitors a temporary path used to store input files. The ASP.NET web page accepts the input files and returns an initial `ExecuteResponse` document, then terminates. When a new input file is detected by the Windows listener service, the delineation tool begins processing the new input files, writing to a log file while processing. Subsequent status update requests from the gateway plug-in access a “`GetStatusUpdate`” ASP.NET web page much like the Linux PHP equivalent. This `GetStatusUpdate` tool parses the log file written by the Delineation Listener service and returns an appropriate `ExecuteResponse` document. If the log file does not yet exist, it is assumed that the Delineation Listener service has not yet begun processing the input, so the `ExecuteResponse` document indicates that the process has been accepted but has not yet begun being processed.

The two ASP.NET web pages (the page handling WPS `GetCapabilities`, `DescribeProcess` and `Execute` requests, and the page handling status updates) along with the Delineation Listener comprise the “WPS Online Plug-in Wrapper” for Windows. Similar to the Linux approach, the same set of web pages can handle multiple different processes by adding a section to the pages for each new process. The Delineation Listener is customized tightly to the delineation process, so with the present architecture a new Windows service needs to be authored for each new process. Future enhancements might include using configuration files for each process as with the Linux approach, and creating a generic Windows listener service for launching actual processing tasks. A generic listener service will eliminate the need to create a completely new listener service for each new process running on Windows.

6 Problems, solutions, and proposed enhancements

Our implementation effort resulted in the identification of six key changes that could improve the WPS protocol. 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 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, browsing for a file of a given type, or other methods for collecting data. The most recent revision of the WPS `DescribeProcess` element has additional metadata that may provide an implementing application with enough data to prompt for input meaningfully; however, the following suggestion may still be useful for allowing an implementing application to behave more intelligently while keeping WPS application independent. In our test implementation, we introduce a new XML element called “`PromptMethod`” to help an application prompt for data. 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 last option, `getmatchingregex`, will accept 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]$$`”. This regular expression would look for the beginning of the input (symbolized by `^`), followed by two letters, from A to Z, independent of case, followed by the end of the input (symbolized by a dollar sign). Some variations for syntax in regular expressions exist due to differing regular expression processing engines [5], hence care should be taken to ensure expressions are designed in such a way that they may be interpreted in the same way on both clients and the server. An example of the suggested addition to WPS is shown in Fig. 1, where the regular expression is defined as “`^\d{8}$$`”. This expression indicates that the start of the string (`^`) should be followed by eight digits (`d{8}`) and the end of the string (`$`), following Microsoft’s regular expression processor syntax. A regular expression can be easily defined as needed, depending on the service and does not require an exhaustive analysis of possible data types during specification design. However, specific XML data types are still useful and needed for core and frequently used data types, particularly for non-string data.

The second suggested change is another addition to the `DescribeProcess` response. Currently there is no mechanism by which a WPS may list the data that is available on the server for use by a given process. An example where this may be useful is in the processes of watershed delineation—defining stream networks and watersheds based on raster elevation datasets [11], [12]. Elevation raster datasets are typically very large, so it is inefficient and often impossible to upload the entire input file to the server, and can be time consuming to transfer the data from another server. Datasets such as elevation data typically do not change often, and may be stored directly on the WPS server to improve performance and efficiency. In our test implementation, 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 Fig. 2. This approach reduces processing by removing the need to upload or transfer the input data, and also creates a means by which a WPS server may act as a data repository as well as processing it and returning results.

Our third suggested change considers that the WPS protocol includes an inconsistency after submitting a job to a server with regards to what should occur next. If the process

```
- <DataInputs>
- <Input>
  <ows:Identifier>inputhuc</ows:Identifier>
  <ows:Title>Input 8-digit HUC to delineate.</ows:Title>
  <ows:Abstract>The input Hydrologic Unit Code (8 digit) for the Automatic Watershed
    Delineation process.</ows:Abstract>
+ <LiteralData>
  <MinimumOccurs>1</MinimumOccurs>
  <PromptMethod>getmatchingregex=^\d{8}$$</PromptMethod>
</Input>
</DataInputs>
- <ProcessOutputs>
- <Output>
  <ows:Identifier>fel</ows:Identifier>
  <ows:Title>Pit Filled Elevation Grid</ows:Title>
  <ows:Abstract>Pit Filled Elevation Grid</ows:Abstract>
- <ComplexOutput defaultFormat="binary/geotiff" defaultEncoding="base64">
```

Fig. 1 Suggested `PromptMethod` element in `DescribeProcess` response

```

<ows:Identifier>inputraster</ows:Identifier>
<ows:Title>Input elevation raster to delineate.</ows:Title>
<ows:Abstract>The name of one of the available data items on this
server.</ows:Abstract>
+ <LiteralData>
- <AvailableData>
  <AvailableDataItem name="Weber Basin NED" desc="Elevation for the
    Weber Basin watershed area." />
  <AvailableDataItem name="Newton Reservoir NED" desc="Elevation data for
    the Newton Reservoir region." />
  <AvailableDataItem name="Idaho NED" desc="Elevation for the state of
    Idaho." />
  <AvailableDataItem name="Utah NED" desc="Elevation for the state of
    Utah." />
</AvailableData>
<MinimumOccurs>1</MinimumOccurs>

```

Fig. 2 Suggested AvailableData element in DescribeProcess response

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 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. Hence we suggest always returning an ExecuteResponse document to provide greater consistency and to simplify client implementation.

After submitting a job to a WPS server, it may be desirable to cancel the requested operation; this eventuality is not supported by the current WPS protocol. 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 needing to cancel a process accesses the URL to trigger a cancellation of the requested process, preventing wasted server processing time.

Our fifth suggested change to the WPS protocol 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. By using the same URL for each request (GetCapabilities, DescribeProcess, and Execute), maintenance and implementation are simplified, whereas confusion or errors can easily arise with differing URLs for these operations.

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 or codes, there is no generic way in which to handle errors on behalf of the user, other than to display the error to the user. A more 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. (Note: The most recent version of the WPS specification includes more options for error handling.)

7 Conclusion

Overall, the WPS proposal was found to be workable as currently designed, and is indeed suitable for many GIS tasks as indicated by our successful tests performing watershed delineation and raster mathematics operations. Implementation of the client component and the server-based WPS processes revealed several opportunities for enhancement including the six specific recommendations provided here. Our test implementation is a simple wrapping approach to expose current and new geoprocessing functionality using the WPS strategy efficiently on the server-side, and it provides a client implementation for an existing open-source GIS platform, MapWindow GIS. This effort should be informative to others working to implement the WPS proposed specification on other GIS platforms, and that our specific test implementation should be immediately beneficial to current MapWindow GIS users. In its present form, our client tool is compatible with version 0.4.0 WPS services (including GML data transfer), and is being updated to function with the newer version 1.0.0 specification.

Any geoprocessing activity may be presented in an “online plug-in” 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 installing updated code. 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 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 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 protocol introduces a standard which will allow diverse developers at various locations to produce geoprocessing offerings and provide them to a variety of client platforms. In these tests, the WPS 0.4.0 protocol has been shown to be workable in its current design. The additional enhancements proposed here should improve WPS implementation and usability significantly.

Acknowledgements This research was funded by the Pacific Northwest Regional Collaboratory as part of a Pacific Northwest National Laboratory project, funded by NASA through Grant No. AGRNXX06AD43G. The authors express gratitude to Trish Mercer for her editing assistance. The manuscript was improved significantly through the feedback from three anonymous reviewers.

References

1. J. Bloomer. Power Programming with RPC. O'Reilly Media: Cambridge, MA, 1992.
2. Environmental Systems Research Institute (ESRI). ArcGIS Desktop Products Data Sheet. WWW document, <http://www.esrichina-bj.cn/produce/esri/arcgisdesktopsheet.pdf>, 2003.
3. Environmental Systems Research Institute (ESRI). ArcGIS 9.2 Webinar—ArcGIS Server: Publishing a Geoprocessing Model. WWW document, <http://events.esri.com/info/index.cfm?fuseaction=seminarRegForm&shownumber=9919> 2006.
4. GIS Competitors Cooperate on OpenGIS Specs. Information Today, Vol. 14(2):15, 1997.
5. Goyvaerts, Jan. Regular Expression Tutorial. WWW document, <http://www.regular-expressions.info/tutorial.html>, 2006.
6. MapWindow Open Source Team. MapWindow GIS 4.0 Open Source Software. WWW document, <http://www.mapwindow.org/>, 2006.
7. M. Neteler. The GRASS GIS software. Presented at a GIS Seminar at Politecnico di Milano, Polo Regionale di Como, November 2006.
8. Open Geospatial Consortium, Inc. Vision and Mission. WWW document, <http://www.opengeospatial.org/about/?page=vision>, 2006.
9. Open Geospatial Consortium, Inc. OpenGIS® Web Coverage Service (WCS). WWW document, <http://www.opengeospatial.org/standards/wcs>, 2007.
10. Open Geospatial Consortium, Inc. OpenGIS® Web Processing Service (WPS) Specification. WWW document, http://portal.opengeospatial.org/files/?artifact_id=24151, 2007.
11. O. Planchon and F. Darboux. “A fast, simple and versatile algorithm to fill the depressions of digital elevation models,” *Catena*, Vol. 46:159–176, 2001.
12. G. Savant, L. Wang, and D. Taux. “Remote Sensing and Geospatial Applications for Watershed Delineation. Conference Proceedings: Integrated Remote Sensing at the Global, Regional and Local Scale.” *IAPRS*, Vol. XXXIV, Part 1, ISSN 1682-1750, 2002.
13. D. Tarboton. Terrain Analysis Using Digital Elevation Models (TauDEM). WWW document, <http://hydrology.neng.usu.edu/taudem/>, 2005.
14. W3C. Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation, 6 October 2000.
15. Initiative. Web Processing Service (WPS). WWW document, http://52north.org/index.php?option=com_projects&task=showProject&id=21&Itemid=127



Christopher D. Michaelis received a Bachelor's degree in computer science from Utah State University in 2005 and a Master's degree in geographic information science from Idaho State University in 2007. His background includes programming for database systems, internet based and client/server communication, and geospatial software. He is currently a software consultant and the lead software developer for the MapWindow project (<http://www.MapWindow.org>) at Idaho State University.



Daniel P. Ames received a Ph.D. in Civil and Environmental Engineering from Utah State University with an emphasis in watershed management using Bayesian decision networks. He also received a Master's degree from Utah State University in the area of time series forecasting using nonlinear methods. He is currently an assistant professor at Idaho State University, where he is the program coordinator for the Master of Science in geographic information science. He is also the project leader for the open source MapWindow GIS (<http://www.MapWindow.org>) project.