IMPLEMENTATION WAYS FOR WEB COVERAGE SERVICE STANDARD

DOI: http://dx.doi.org/10.18509/GBP.2016.01
UDC: 004.774.6

Evgeny Panidi
Anton Terekhov
Mark Mukhamedzyanov
Saint-Petersburg State University, **Russia**

ABSTRACT

This study is dedicated to design and implementation of two software prototypes, which are facilitate fast and simple Web publication of the raster coverages without specialized dedicated Web infrastructure. First prototype is implemented in Python programming language as a server-side Common Gateway Interface (CGI) application. The second is implemented as a module for Node.js platform, which is very popular for development of the multipurpose Web applications. Both solutions can be deployed using virtual shared hosting. This feature expands the opportunities of geospatial data publication on the Web for small- and medium-scale projects.

Keywords: Raster Coverage, Web Coverage Service, Geospatial Data Web Publication, Open Geospatial Standards, Free and Open Source Software for Geospatial

INTRODUCTION

Web publication of the geospatial data and software tools is a modern and effective way for distribution of the geospatial data, as well as for multiuser collaboration in the area of formation and use of these data. A couple of international open standards developed by Open Geospatial Consortium (*http://www.opengeospatial.org/*) to unify Web publication techniques and schemas in the case of geospatial resources. However, many issues remain unresolved in this area. First, it is the issue of effective implementation of the server-side software and platforms, which are used for publication of the geospatial data on the Web. One of aspects of the effectiveness in this case is related to that the most of the currently developed and available software are designed for use of the dedicated servers or cloud infrastructures. This feature limits seriously the capabilities of geospatial data integration into already developed geospatial Web resources that hosted on non-dedicated hosting.
Our previous studies related to increase of flexibility of the Web-based geospatial data processing techniques. In these studies, we investigated the opportunities of client resources use for processing data with the help of geospatial data processing Web services [8], [10]. One of previously produced conclusions, was that the geospatial Web service publication schemas (both processing and data services) should be simplified in the case of small- and medium-scale projects [11]. The conclusion was caused by abovementioned complexity of the existing geospatial Web publication platforms. Currently, we have investigated some possibilities of lightweight Web publication of the raster coverages, i.e. use of publication tools that compatible with common software platforms for Web publication. Raster coverage is one of the highly demanded data type for geospatial analysis in many thematic areas. Nevertheless, the OGC Web Coverage Service (WCS)

standard [1] implemented currently only in the specialized complex server-side software platforms (i.e., Web mapping servers), which assumes use of the dedicated servers.

TOOLS AND TECHNIQUES

OGC WCS standard: The WCS standard incorporates the Core specification [1] and number of extensions [2], [3], which regulate the order of additional transformations applied to raster coverages before they are uploaded to the client. Core specification assumes only three types of requests: GetCapabilities, DescribeCoverage, and GetCoverage. The first one should be responded with an XML structure containing the metadata of the Web service. Simultaneously, the second responded with an XML structure containing metadata of the selected raster coverage. Finally, the third allows to request loading of the selected coverage.
Additionally, the Core specification, involves subsetting (using some bounding box parameters parameters) before coverage uploading. In particular, the Trim operation allows to retrieve a coverage sub-area using some bounding rectangle, and the Slice operation allows to apply the dimensions reduction to the coverage (i.e., reducing of the number of raster bands, in terms of raster data model).
Architecture: Regardless of the programming pattern, implementation of the WCS server assumes the presence of three internal logical modules, in case of the Core functionality implementation. The first is program interface module (HTTP Module), which incorporates all of the program functions and methods applied for handling of the incoming requests and generation of the server responses in accordance with the OGC WCS specifications. In both cases, the Hyper Text Transfer Protocol (*https://www.w3.org/Protocols/rfc2616/rfc2616*) is used. The second is Metadata Management Module that incorporates functions and methods applied to read, write, and transformation of the coverage metadata, regarding to the coverage store format. The third is Coverage Management Module that allows to implement the Trim and Slice operations support.
Our approach to implementation of the WCS standard specifies two basic constraints. These are the enablement of coverage publication on the common hosting (i.e., virtual shared web hosting that do not provide any capabilities to install additional server-side applications into the server operating system), and the enablement of WCS server portability (i.e., capability of deployment via simple copying).
Programming tools: Support of the WCS specifications implies the need to handle the HTTP requests of the specialized types and to perform server-side operations related to management and processing of the coverage data and metadata (including subsetting operations and uploading data to the client). Currently, all of the widely known WCS servers are incorporated into the different server-side geospatial platforms (GeoServer, MapServer, etc.) that cannot be deployed on virtual shared hosting. Therefore, the deployment constraints we established raise the need to develop a compact easy deployable Web application.
It is possible to allocate two design approaches to the Web applications publication on virtual shared Web hosting. The first one involves the capability to develop external applications, and the subsequent connect of the applications to the Web server and Web interface via a program interface (e.g., Common Gateway Iinterface (CGI) applications, Java servlets, Application Programming Interfaces (APIs) of different software Web

servers). An alternative approach assumes the development of the applications that deeply integrated with the Web interface. In this case, different backend platforms and runtime environments can be used, which also integrate some development tools (e.g., Hypertext Preprocessor (PHP), Ruby on Rails, Node.js, etc.).
In our study, we used both approaches. To implement the CGI-based WCS server prototype, the Python 2.7 programming language was used (*https://www.python.org/*), which is widely demanded now in scientific research [9]. Its popularity also continues to grow in geospatial studies. In this case, the choice of Web hosting has the only limitation, which is support of the Python CGI scripts execution.
To implement the prototype of integrated WCS server, the Node.js (*https://nodejs.org/*) platform was used, which is a framework for Web applications development and a JavaScript runtime environment. Node.js is a rapidly developing technology with a fast-growing community. Current versions of the Node.js are equal in performance to other popular backend technologies (e.g., Java, Python, Ruby, PHP, etc.). More and more companies offer virtual shared Web hosting for Node.js applications (or more exactly, for Node.js extensions).

CGI-BASED WCS SERVER CASE

Through the CGI, the CGI application can be executed on the server upon client's request. The application response is returned to the client also through this interface. The response of the WCS server should be either a special way structured (in accordance with the WCS standard) XML object or collection of geospatial datasets. Our Python CGI WCS server handles incoming requests and generates corresponding responses using internal HTTP Module (Fig. 1).

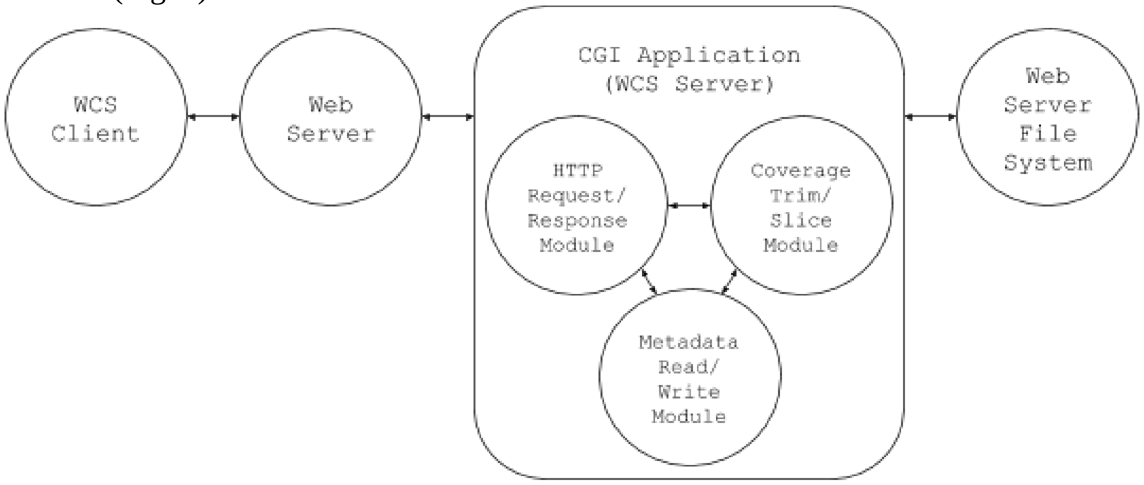
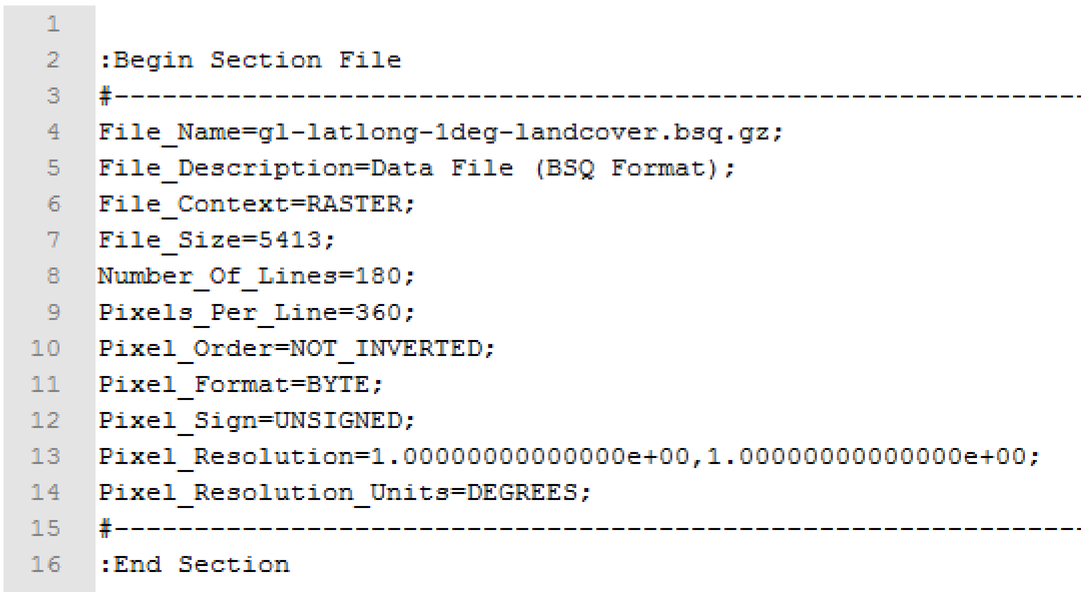

WCS Client
Web Server
CGI Application (WCS Server)
HTTP Request/ Response Module
Coverage Trim/ Slice Module
Metadata Read/ Write Module
Web Server File System

Figure 1. General structure and interactions of the Python CGI-based WCS server.

It should be taken into account, that quite a large number of formats is used today to store raster coverages. Some formats involve storing metadata incorporated into one file with the coverage data (e.g., GeoTIFF format [12]), and some split data and metadata into separate files (e.g., Generic Binary Band Sequential format – BSQ/HDR [5]). We used two mentioned formats to model both approaches. The BSQ/HDR format was used for the implementation of CGI-based WCS server, and the GeoTIFF format for the implementation of Node.js-based WCS server (see below). However, the implementation of the support for any other raster coverage format can be brought to one of these two models.

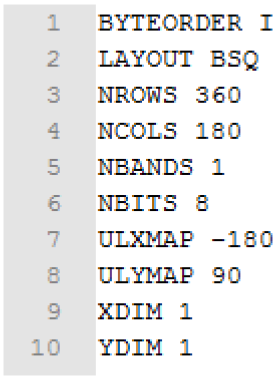
Metadata management is performed by the Metadata Management Module (Fig. 1). The BSQ/HDR format involves the raster data storage in the form of a pure binary array without a header. In this case, the data of every raster band are written to a file sequentially one band after another. The metadata is stored in the external header text file (HDR file [5]). In some cases, metadata may be stored in the text file that formed using another notation. In our case, this implies that it can be needed to generate HDR file additionally. As an example of such notation, the metadata of the AVHRR Global Land Cover datasets (*http://glcf.umd.edu/data/landcover/data.shtml*) can be mentioned (Fig. 2), which we can convert quite simple to the HDR representation (Fig. 3).



```

:Begin Section File
#---------------------------------------------------------------
File_Name=gl-latlong-1deg-landcover.bsq.gz;
File_Description=Data File (BSQ Format);
File_Context=RASTER;
File_Size=5413;
Number_Of_Lines=180;
Pixels_Per_Line=360;
Pixel_Order=NOT_INVERTED;
Pixel_Format=BYTE;
Pixel_Sign=UNSIGNED;
Pixel_Resolution=1.00000000000000e+00,1.00000000000000e+00;
Pixel_Resolution_Units=DEGREES;
#---------------------------------------------------------------
:End Section
```

Figure 2. A fragment of text representation of the AVHRR Global Land Cover metadata.



```
 1  BYTEORDER I
 2  LAYOUT BSQ
 3  NROWS 360
 4  NCOLS 180
 5  NBANDS 1
 6  NBITS 8
 7  ULXMAP -180
 8  ULYMAP 90
 9  XDIM 1
10  YDIM 1
```

Figure 3. Content of the HDR file.

It should be noted, that the necessary and sufficient information needed to ensure the functionality of the WCS Core are the coordinates of the initial raster cell (lines 7 and 8, Fig. 3) and the raster spatial resolution along the axes (lines 9 and 10, Fig. 3).
The functionality of Coverage Management Module (Fig. 1) is applied to produce Trim and Slice operations. It was necessary to ensure the possibility of the CGI WCS server implementation as a standalone Web application that does not require any dependencies and installations of additional libraries or applications (which are not possible on virtual shared hosting). Regarding this, we had the key problem that consisted in the inability to

manage raster formats using the GDAL library (*http://www.gdal.org/*). This library is the de facto standard for management of geospatial raster data (i.e., read, write, convert, extract metadata, etc.). GDAL was ported to several programming languages, including Python. However, the library needs to be installed in the operating system.

Due to the unavailability of the common raster data management instruments, we have implemented the obvious solution. We developed methods for native Python metadata parsing and saving from/to HDR text file as well as native Python reading and writing of the coverage binary data from/to BSQ file.

Thus, to publish raster coverage on some Web resource using developed CGI WCS server prototype, the administrator of the Web resource needs to do three steps. He needs to upload CGI application (WCS server) files into the CGI folder on the server; then, to upload the coverage file and the metadata file into a dedicated folder on the server; and finally, to specify the path to this folder in the configuration file of the CGI application.

When a client requests the list of available coverages from the WCS server (executes GetCapabilities request), the WCS server search all of the files of a supported format, and executes XML response. Further, when client requests metadata of a particular coverage (DescribeCoverage request), the server reads the coverage metadata, and executes another XML response. Finally, responding to the GetCoverage request, the WCS server reads the data from the selected coverage file, and generates a derivative coverage file, using Trim and Slice operations regarding to the request parameters. Then, produced coverage is uploaded to the client.

Additionally, the WCS server supports the use of manually generated metadata files that contain the WCS metadata and metadata of the separate coverages in XML format. In the case of such files presence in the data folder, automatic generation of metadata for server is not produced.

NODE.JS-BASED WCS SERVER CASE

Architecture of the Node.js platform provides flexible functionality extension through the connection of additional modules, which are installed on the platform using the integrated deployment mechanism [4]. The platform itself and any additional modules are distributed as Free and Open Source Software (FOSS). To implement support of needed HTTP requests and responses we used the Express.js framework (*http://expressjs.com/*), which extends the Node.js functionality through the introduction of basic methods for receiving of the HTTP requests and responding to them [7].

Using Express.js we implemented handling of the basic WCS requests (GetCapabilities, DescribeCoverage, GetCoverage). Three eponymous methods were developed in addition to the basic Express.js functionality. Thus, Express.js extended with new methods performs the role of the HTTP Module (Fig. 4). As in the case of CGI-based WCS server prototype, the requests are executed using the HTTP GET method. Next URL schema is used for the request execution:

<host>:<port>/<path>/<method>?<parameters>

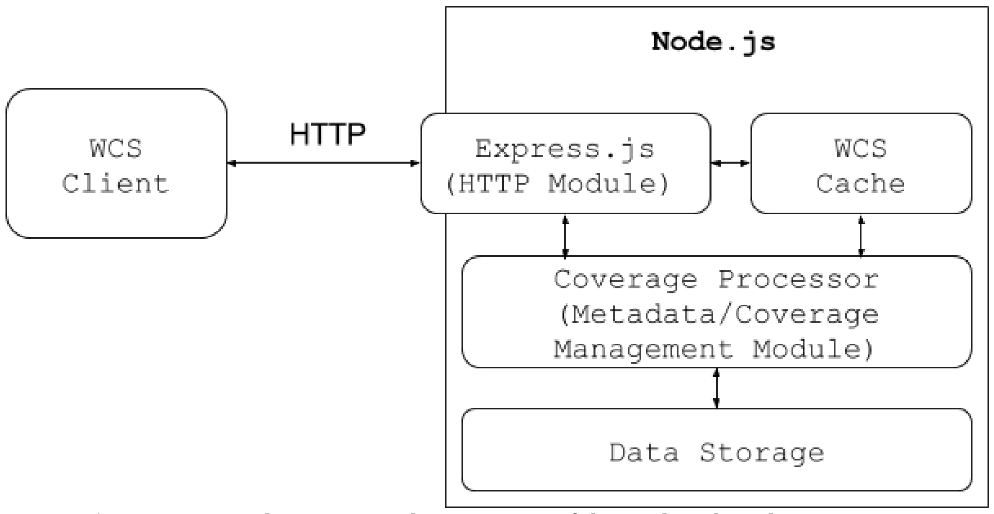


Figure 4. General structure and interactions of the Node.js-based WCS server.

As it was mentioned above, our Node.js-based WCS server prototype supports GeoTIFF format for coverages storing. To implement GeoTIFF support, we used the GeotiffParser.js JavaScript library (*https://github.com/xlhomme/GeotiffParser.js*), which is designed only for reading GeoTIFF files. To implement the Trim and Slice operations and possibility of writing the coverage into new file, we have expanded the library with additional methods. Thus, in this case, the extended GeotiffParser.js library plays the role of a Metadata Management Module and Coverage Management Module, at the same time.

Storing of the published coverages is carried out in server file system. However, control and management of the Data Storage is implemented directly in Node.js. The administrator needs only to upload the coverage files into the dedicated coverages folder on the server. When the GetCoverage request is handled, the server loads the selected coverage file into RAM; checks the validity of the Trim/Slice bounding box parameters that were received with the request; and copies the trimmed/sliced part of the original coverage into the new GeoTIFF file, which is temporarily cached and uploaded in response to the client.

Because the Node.js platform and JavaScript in general are executed in asynchronous mode, the WCS server unable to handle other incoming requests when running the coverage subsetting. To solve this problem, we used processing with interruptions. At the every fixed computation step (in looping operations), the application stops and verifies the presence of the incoming requests. If any new incoming request is presented, it is logged and placed into the processing stack. Then the interrupted processing loop continues.

In the case when another request estimated to be handled faster (for example when the bounding box is smaller), the previous request handling is suspended, and new handling starts. Due to this technique, we obtain a self-balancing option in the WCS server functionality.

CURRENT RESULTS AND FUTURE WORK

Both software prototypes developed in the context of described study are licensed as the FOSS. Python CGI WCS executable instance and its source code are accessible at *http://195.70.211.131/pywcs/*. The Node.js WCS instance and source code are accessible at *http://195.70.211.131/nodejs/*.

To provide preliminary performance estimation for the developed prototypes, we conducted the tests of the subsetting time (Fig. 5 and Fig. 6). These tests were performed with 3 Gb RAM and 2.2 GHz dual-core CPU. One-channel raster was used in both cases.

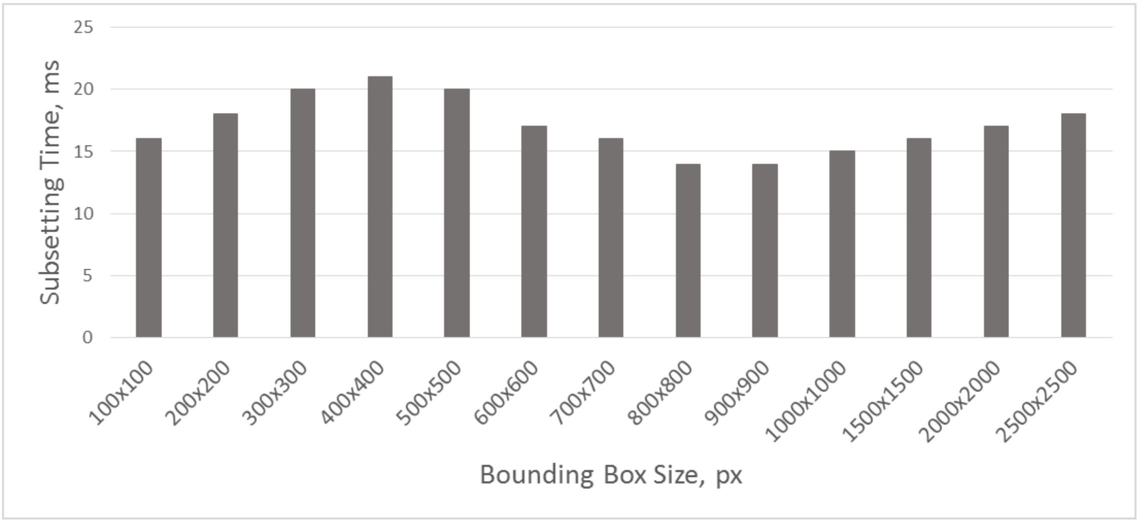


Figure 5. Processing time diagram for the implemented Python CGI-based WCS server.

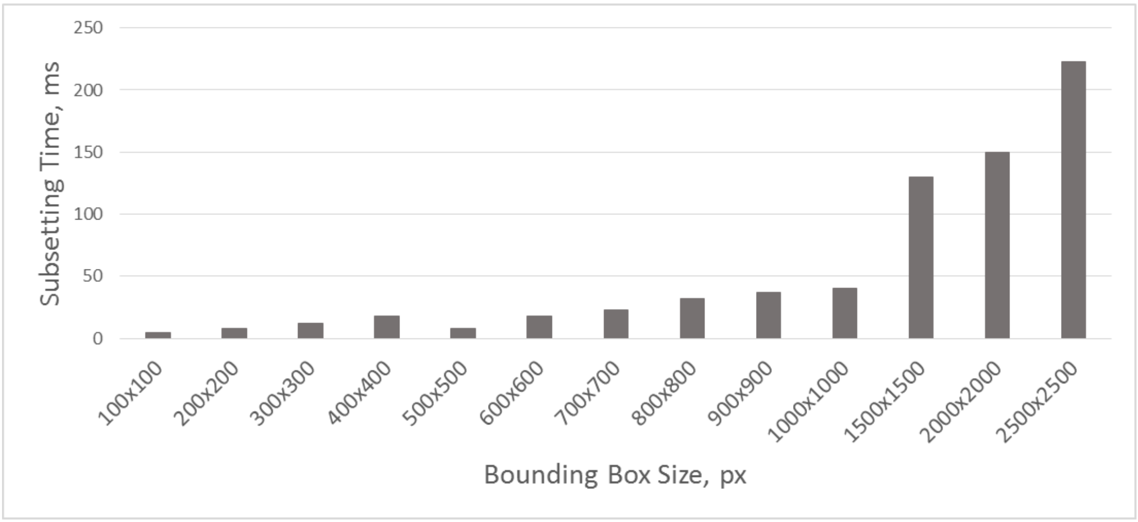


Figure 6. Processing time diagram for the implemented Node.js-based WCS server.

Presented graphs show the time that were spent for extracting needed segments of the raster coverage, when the coverage is previously loaded into RAM. Significant differences in the processing time is caused by the differences in the program code design. In the case of Python CGI, the coverage data is extracted as the row segments from the coverage binary array, while in the case of Node.js, the data is extracted as separated coverage cell values (due to design of theGeotiffParser.js library [6]). This technique was used as a temporary solution, to facilitate development of the application prototype. However, subsetting time measured in the case of Node.js for the sub-coverages up to 500x500 pixels demonstrates good potential of computations optimization.

Future work implies the study of implementation techniques for extensions of the WCS standard, both for the CGI case and for the Node.js case. However, the actual issue is the computational performance raising, through the source code optimization or through the use of some additional techniques. For example, the possibility of processing clustering can be explored for the case of multiple CPU cores availability. Another potentially interesting approach can be a breaking the stored coverage into parts (tiles or separated bands) to optimize the process of coverage subsetting.

REFERENCES

[1] Baumann P. (ed.), 2012. OGC WCS 2.0 Interface Standard – Core: Corrigendum. OGC 09-110r4. Version: 2.0.1, 2012-07-12

[2] Baumann P., Yu J. (eds.), 2014. OGC Web Coverage Service Interface Standard – CRS Extension. OGC 11-053r1. Version: 1.0, 2014-03-11

[3] Baumann P., Yu J. (eds.), 2014. OGC Web Coverage Service Interface Standard – Scaling Extension. OGC 12-039. Version: 1.0, 2014-02-26

[4] Casciaro M. Node.js Design Patterns. Packt Publishing, 2014, 454 p.

[5] Extendable Image Formats for ArcView GIS 3.1 and 3.2. ESRI White Paper, July 1999, 19 p.

[6] GeotiffParser.js source code. Accessible at https://github.com/xlhomme/GeotiffParser.js/blob/master/js/GeotiffParser.js (Visiting date is February 24, 2016).

[7] Hahn E. Express.js in Action. Manning Publications, 2016, 245 p.

[8] Kazakov E., Terekhov A., Kapralov E., Panidi E. WPS-based technology for client-side remote sensing data processing. International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XL-7, Issue W3, 2015, pp. 643-649. doi:10.5194/isprsarchives-XL-7-W3-643-2015

[9] Martelli A., Ravenscroft A., Ascher D. Python Cookbook. 2nd Edition, O'Reilly Media, 2005, 846 p.

[10] Panidi E., Kazakov E., Kapralov E., Terekhov A. Hybrid geoprocessing Web services. SGEM2015 Conference Proceedings, Book 2, Volume 1, 2015, pp. 669-676. doi:10.5593/SGEM2015/B21/S8.084

[11] Panidi E., Terekhov A., Kapralov E., Kazakov E. Case study of lightweight geospatial web servers' implementation. International Symposium on Digital Earth, October 5-9, 2015, Halifax, Nova Scotia, Canada, Abstract Volume, 2015, p. 49.

[12] Ritter N., Ruth M. GeoTIFF Format Specification. GeoTIFF Revision 1.0. Specification Version: 1.8.2, 2000.