A Web Processing Service for GRASS GIS to Provide on-line Generalisation

Wolf Bergenheim, L. Tiina Sarjakoski, Tapani Sarjakoski
Finnish Geodetic Institute, Department of Geoinformatics and Cartography

ABSTRACT

Over the last decade, considerable steps have been taken to improve methods for generalisation. Recently, research on Web services has been intensively aiming to provide solutions also for online generalisation. At the same time, defining and implementing map generalisation as a Web service has gained a lot of interest among the research community. The aim of this paper is to describe an implementation of a real-time generalisation service that can be accessed over the Internet. In this paper, we describe an experimental implementation of the new Open Geospatial Consortium Web Processing Service that we call the "*WPS PHP Server*". The service is built over an existing Geographic Information System platform, the Geographic Resources Analysis Support System (GRASS). A case study that describes how to use the WPS PHP Server to generalise roads on the fly is presented. Finally, the implementation is compared to existing ones with suggestions for future work.

1. INTRODUCTION

During the past few years, there has been a growing interest for defining and implementing map generalisation as a Web service. One of the turning points of those efforts has been the workshop of the ICA Commission on Generalisation and Multiple Representation, held in August 2007 in Moscow (ICA 2007), during which interoperable Web generalisation service specification was identified as one of the key issues for further research and development. Later the same year, a working group meeting was held to pursue this research line (Foerster et al., 2008). The research reported in this paper is one branch of that activity. In this paper, our approach is strongly experimental. We describe an implementation of the new Open Geospatial Consortium (OGC) Web Processing Service (WPS). We call the implementation "*WPS PHP Server*", since it is a WPS implemented in the PHP[1] scripting language. The OGC Web Processing Service (WPS) interface standard provides rules for standardising inputs and outputs (requests and responses) for geospatial processing services. The standard also defines how a client can request the execution of a process and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and the clients' discovery of and binding to those processes. The data required by a server implementing the WPS specification can be delivered across a network, or it can be available at the server, (OGC, 2007).

We chose to build a service over an existing Geographic Information System platform (GIS), the Geographic Resources Analysis Support System, commonly referred to as GRASS GIS, to show how the generic WPS interface can be used to build a specific tool. Since open source software is being used more often, especially in scientific research, our implementation, WPS PHP Server, is also built by using only open source tools and environments.

[1] PHP is a recursive acronym of "PHP: Hypertext Preprocessor" (RecAcro, 2009)

1.1 Background

The GRASS GIS is used for geospatial data management and analysis, image processing, graphics/maps production, spatial modelling, and visualisation. GRASS GIS is currently used in academic and commercial settings around the world, as well as by many governmental agencies and environmental consulting companies. GRASS GIS is an official project of the Open Source Geospatial Foundation (OSGeo), (GrassGIS 2009).

The aim of this paper is to describe an implementation of a real-time generalisation service that is accessible over the Internet. Recently, a generalisation module, called *v.generalize*, has been added to GRASS GIS (Bundala and Bergenheim 2007). This generalisation module supports multiple line simplification and smoothing operations. One of these is the Douglas-Peucker line simplification algorithm (Douglas and Peucker 1973). The GRASS GIS is, however, only a desktop GIS, but by design it can be separated from its graphical user-interface (GUI). Actually, the default GRASS GIS GUI is a wrapper around the independent modules that make up the GRASS GIS. The modules themselves are independent of the graphical user interface (Neteler 2008). This is exactly what the WPS PHP Server does, it wraps around GRASS GIS to provide access to the new generalisation module via the OGC WPS interface. We found that by designing our implementation, the WPS PHP Server, to be flexible, it would be easy to allow other processes to be exposed in this way.

In the next section, we briefly discuss the previous research related to online generalisation and Web services. In Section 3, we present the architecture of our implementation, the WPS PHP Server, in detail. Section 4 describes a case study that shows how we use the WPS PHP Server to generalise roads on the fly. In Section 5, we compare our implementation to existing ones and discuss future work. Finally, we present our conclusions.

2. PREVIOUS RESEARCH

Over the last decade, considerable steps have been taken to improve methods for generalisation. Recently, research on Web services has been intensive aiming to provide solutions also for online generalisation (Lehto and Kilpeläinen 2000, 2005; Harrie and Johansson 2003; Sarjakoski et al. 2005; Sarjakoski and Sarjakoski 2007). Edwardes et al. (2003) proposed a common research platform for map generalisation technology that is based on open source frameworks and standards. Sarjakoski et al. (2005) presented work towards a standardised Web generalisation service for real-time generalisation. The access interfaces of the service were based on the Open Geospatial Consortium Specifications. Lehto (2007) studied real-time map generalisation in a broader context of real-time content transformations in Web service-based delivery architectures.

Edwardes et al. (2007) presented experiments on building an open generalisation system. They presented two open architectures for providing access to generalisation processing. The first one takes the approach of a middleware component that presents generalisation functionality through a crude, Web mapping interface and is based on open source technologies for its core infrastructure. The other one adopts the model of a Web-service registry that provides access to generalisation operators in a distributed and platform-independent way. The registry model implements a set of Web service standards that allow generalisation researchers to register and expose their operations. Foerster et al. (2008) reported about the working group status and discussed an interoperable Web generalisation services framework and technical requirements of the service.

Although several prototype implementations on Web generalisation services have been presented and discussed, an approach in which the service is built upon an existing open source GIS platform, such as GRASS GIS, does not seem to exist. Therefore, in our approach, we decided to focus on such an approach.

3. ARCHITECTURE OF THE SERVICE

Our generalisation service implementation presented in this paper, the WPS PHP Server, is written in the PHP Web-scripting language. PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl, with some unique PHP-specific features. The goal of the language is to allow Web developers to write dynamically generated pages quickly (PHP). The system is designed using an object-oriented programming model, and taking advantage of the new object-oriented features of version 5 of the PHP scripting language. We run the PHP system as a module of the Apache Web server, which means that the Apache Web server deals with the network traffic and starts the PHP script. The PHP scripting environment initialises some global variables with the parameters of the HTTP request (GET variables) and assigns a file stream to any input POST data, which makes reading data sent as data in a POST request as easy as reading from a file. Only after this, the PHP parser starts reading our software. Firstly, our implementation needs to initialise itself. This initialisation happens for each incoming request, since HTTP requests are stateless. Each request is, thus, totally separate and needs to load some initial data. The initialisation is quite straightforward. In the beginning of the initialisation, the configuration file, called configuration.xml, is located and parsed. This XML file contains information that tells the system where it can find the environment (like a working directory, GRASS GIS setup, etc.). Figures 1 – 6 are all drawn using the Unified Markup Language (UML). Figure 1 shows the sequence of the initialisation in a UML sequence diagram, but leaves out the module initialisation, which is discussed below.

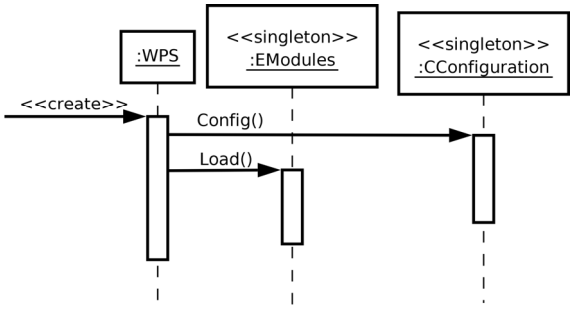

Figure 1: A UML diagram showing the Initialisation sequence.

There are two HTTP methods used to access the WPS services: GET and POST (OGC, 2007). Each input method has a handler class to parse the input. The GET handler, called GetParser, fetches the key and value pairs of variables that are embedded in the URL of the request (RFC, 1999). The PostParser parses the XML sent as a POST document for the input. Both classes inherit from a common base class called InputParser. Figure 2 shows the relationship between these parser classes, in UML notation.

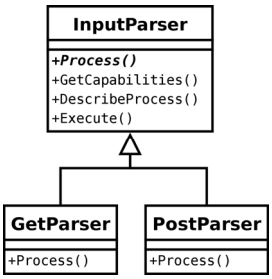

Figure 2: A UML class diagram of the input parsers.

3.1 WPS GetCapabilities request

Figure 3 shows the UML collaboration diagram of the objects in the system during a GetCapabilites request. In the figure, we can see that the object of class WPS calls the Process method of the mParser object (of class GetParser), that in turn calls the GetCapabilites method of its parent class, InputParser. That method determines which sections of the GetCapabilities document are requested. The returned GetCapabilities document contains only the sections that were requested. If the client is requesting the "ALL" section, then the full GetCapabilites document is returned. If the ProcessOfferings section is needed, the GetCapabilities method is called for all objects of the EModule class. Each of these objects in turn calls the BriefDescription method of each EProcess that they govern.

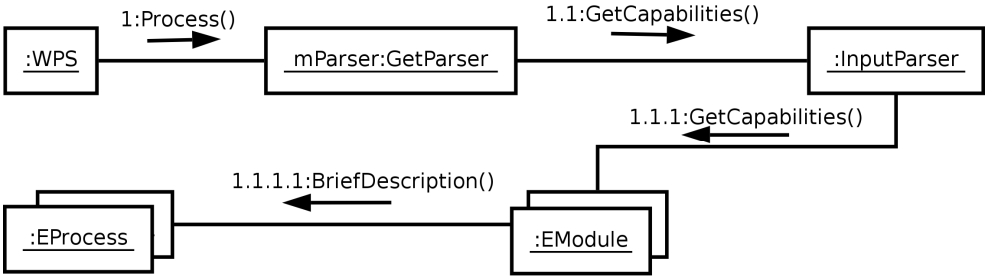

Figure 3: A UML collaboration diagram showing the WPS GetCapabilities request processing.

3.2 WPS ProcessDescription request

Figure 4 shows the UML collaboration diagram of the objects in the system during a ProcessDescription request. Since the ProcessDescription can be requested either by a GET or a POST method (OGC, 1999), the WPS object needs to create either a GetParser object, if the method was via a HTTP GET request, or a PostParser, if the request was via a HTTP POST request. The GetParser parses the Identifiers as key-value pairs from the query string of the GET request into an array. The POST parser builds a DOM model of the XML sent as input of the request and builds an array of Identifiers from the key-value pairs contained in the XML. The mParser object (that is either a GetParser or a PostParser) calls the DescribeProcess method of the common super class, InputParser. The DescribeProcess method calls a ProcessDescription method for all modules. This method returns a list of ProcessDescriptions that match. After all Identifiers and all modules have been gone through, the ProcessDescription document is returned.

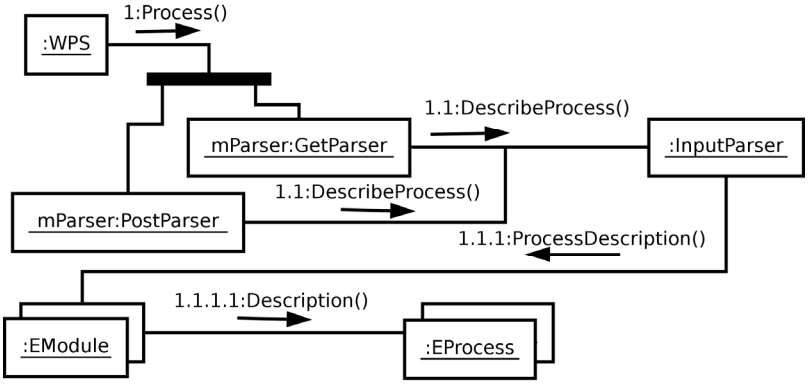

Figure 4: A UML collaboration diagram showing the WPS ProcessDescription request processing.

3.3 WPS Execute Request

Our implementation, the WPS PHP Server, only supports Execute requests via a HTTP POST operation. Even though both GET and POST methods are defined in the OGC 05-007r7 specification, the GET method is optional. Figure 5 shows the UML collaboration diagram of objects in the system during an Execute request. The sent ExecuteRequest document is received by the main object that is of the WPS class. This object creates an object of PostParser class (mParser) and calls the Process method of it. The Process method parses the ExecuteRequest document. The mParser object then calls the Execute method of the super class InputParser. Any input maps sent by reference are fetched at this point. The Execute method of the InputParser class calls the Execute method on all modules until it gets a non-false return value, meaning that the correct module has been found, and the process has been executed. Each module checks if it knows of a process with the given Identifier. If it does not, then it returns false. If it does, it calls the Execute method of the process, and returns the result. Once the process execution is complete, the Execute method of the InputParser class constructs an Execute response message that contains the resulting maps in-line (default) or as a reference, depending on what the client has requested.

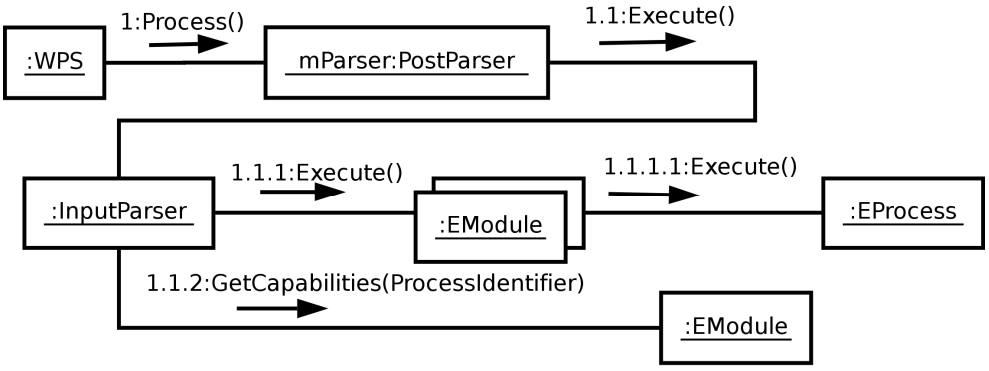

Figure 5: A UML collaboration diagram showing the WPS Execute request processing.

3.4 Modules

A module in the context of the WPS PHP Server is simply a set of processes, grouped into a single unit. A module can, for instance, contain processes that are related or similar, or maintained by the same person. In other words, it is simply a convenience tool. Each module is defined by an XML file that is read when the WPS PHP Server initialises. The module initialisation proceeds as follows: The system reads each XML file found in the modules directory. Such a file consists of multiple process elements that define a single process. Each file can, thus, define multiple processes. For each process element found in the file, an object of either EGrassProcess or ENativeProcess is created. Both of these are sub-classes of EProcess that contain common methods and serves as the interface to processes. Each module is, thus, represented by an EModule class in the WPS PHP Server. The processes are stored as an array of EProcess objects. Figure 6 shows the relationships between processes and modules as a UML class diagram.

In order to add a process, no programming at all is needed. The only thing one has to do is to write an XML file and connect a process to a GRASS GIS module (or another program). For more complex processes that require chaining multiple GRASS GIS modules, a script or a program needs to be written, however, there is no limitation of what programming language can be used or even should be used.

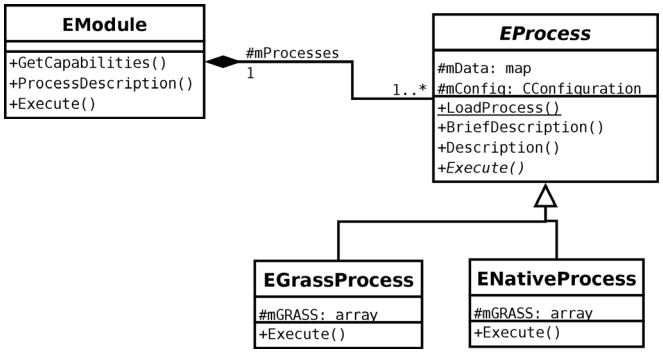

Figure 6: A UML Class diagram showing the relationship between modules and processes

3.5 Processes

There are two kinds of processes in the WPS PHP Server: GRASS and native processes. The GRASS processes, represented by the EProcess class, are processes that use GRASS GIS to do the work. Usually, it is a single GRASS GIS module, but it can also be a script (to chain multiple modules) or other program that runs within GRASS GIS. The native processes, represented by the ENative class, do not use GRASS GIS for data processing. The GRASS processes are separated because there are two extra steps needed to make a GRASS process run. First, the transferred (or referenced) data needs to be imported into a GRASS GIS database. After processing is done, the result needs to be exported from the GRASS GIS database into a format suitable for return. At the moment, only the Geography Markup Language (GML) format is supported. The actual program that does the work (in either type of process) can be any kind of executable format that the operating system can run.

4. A CASE STUDY IMPLEMENTATION

Our case study implementation shows how to define in the WPS PHP Server a WPS-interface for the v.generalize module in GRASS. We also demonstrate its usage for on-demand generalisation. As a client, we use an in-house, WPS OpenLayers client extension. OpenLayers is an open source project from OSGeo. OpenLayers makes it easy to put a dynamic map in any Web page. It can display map tiles and markers loaded from any WPS and WMS source (OpenLayers 2008).

In our case-study, we use the Finnish Digiroad data in order to test the generalisation of road lines. Digiroad is a national road and street database that contains information on the geometry of roads and streets as well as their physical properties (Digiroad 2009).

We have defined three generalisation processes each of which uses the v.generalize module. All three processes use the Douglas-Peucker line simplification algorithm, but they have different threshold values. Figure 7 shows the Digiroad data before generalisation, and Figure 8 shows it after generalisation by the simplification operator. The GRASS GIS module v.generalize implements a version of the Douglas-Peucker algorithms that only promises to keep the end-points intact; thus, intersections may move, and self-intersection can occur. However, GRASS provides a tool to break poly lines at intersections. This tool could be placed in the generalisation process so that the poly lines would first be broken at intersections. That would then fix the location of the intersections.

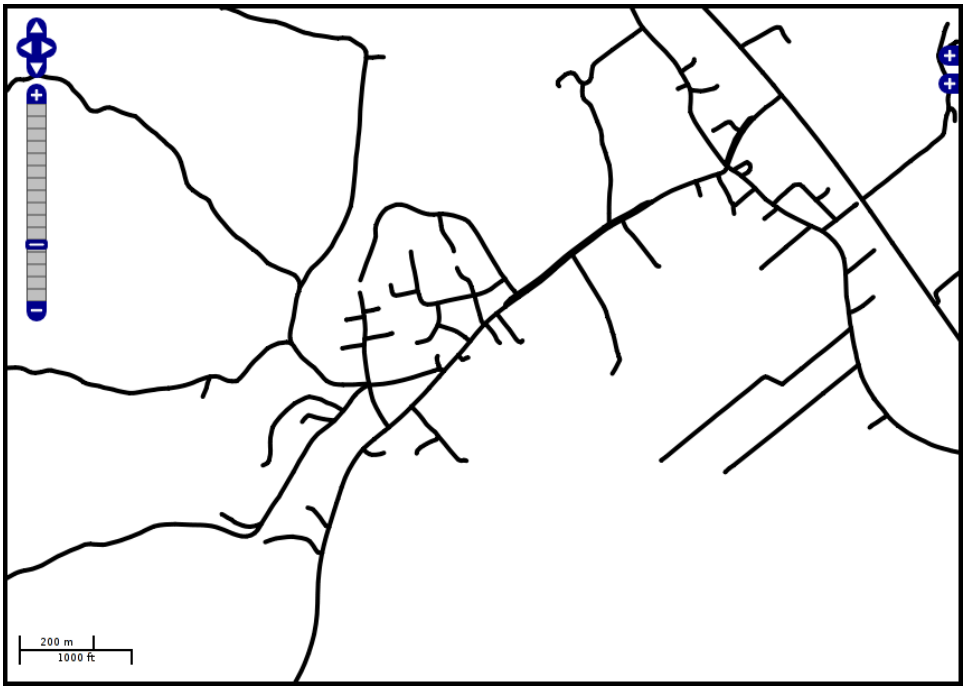

Figure 7: The Digiroad data, before generalisation.

Figure 8: The Digiroad data after generalisation with the simplification operator.

5. DISCUSSION AND FUTURE WORK

The GRASS GIS uses the OGR Simple Features Library to export data into multiple data formats, including GML. Unfortunately, the GML generated by the OGR Simple Features Library is not really valid and requires a certain amount of post-processing. However, because the OGR Simple Features Library supports multiple data formats, the WPS PHP Server could easily be made to support other formats, such as GeoJSON or Keyhole Markup Language (KML) and others.

The support for raster maps should also be implemented in future work. The GRASS GIS uses the Geospatial Data Abstraction Library (GDAL) to provide support for multiple raster formats. This means that raster data support could be added with relatively little effort.

The support for multiple input and output maps also needs to be developed. Many processes, like intersection, are such that they take two maps as input and produce a single output map. There might also be processes that produce multiple output maps from a single input map, so it is quite important that multiple input and output maps are supported.

Our implementation, the WPS PHP Server, should also be improved to support the HTTP GET method for the Execute operation, as specified in the OGC 05-007r7 specification

There is basic support for non-GRASS processes. This support could be further developed to allow a wider variety of GIS software to be given a WPS interface. The problem is the command line interface. All GRASS GIS modules have a similar command line structure because of the design of GRASS GIS (Neteler 2008) and can thus all be easily encapsulated. The command line parameters for native processes need to be defined more formally, so that wrapping the commands can be done in a reasonable way. A developer's guide would need to be written to give developers a better idea of what the WPS PHP Server expects. The existing software would probably need some wrapper script to work (if the source code is not available).

As discussed above, other implementations of the OGC WPS specification have also been developed. However, we feel that our implementation, the WPS PHP Server, is more flexible in many ways. The WPS server from 52° North (52 North 2008), 52n WPS, is an open source project. The 52n WPS is written in Java and, as such, requires that processes are written in Java. The 52n WPS server does not support GRASS GIS, so processes need to be developed for it. There is, however, an experimental plug-in for the Java Unified Mapping Platform (JUMP) and uDig that works in a similar way to our implementation.

Another open source WPS server, PyWPS, is written in Python, a popular scripting language. PyWPS also has native support for GRASS GIS, but it requires a bit of Python programming, where the WPS PHP server only requires an XML file to be written.

The module structure in our implementation, the WPS PHP Server, allows for access control of the WPS PHP Server processes, which means that a single instance of the WPS PHP Server needs to be running to serve processes that multiple people are responsible for. The access control is achieved by setting access control provided by the operating system on the XML files that define the modules. Access permissions can be set so that only the person(s) who should be allowed to modify a module (and the processes it offers) can do that. This access control could be developed further so that the modules would be accessible only after proper authentication. The module structure is also a way to group related processes into more manageable units. One could, for instance, group the services so that each grass module is covered by a WPS PHP Server module, since many GRASS modules have multiple functions (like v.generalize).

6. CONCLUSIONS

The implementation of a real-time generalisation service, described in this paper, shows how the OGC WPS specification can be applied to an existing GIS solution and provide a Web interface allowing remote access not only to GIS data but also to GIS processing. This is potentially useful for application in handheld and other devices with limited computing power, typically used in field work. Wrapping a WPS server over an existing GIS platform saves the amount of work that would normally be needed for developing the processes. If new processes need to be developed they can be developed for the existing GIS platform.

The WPS PHP Server can, at the moment, only process vector maps. The primary goal of this work was to create a WPS server that could do real time generalisation of road and building data. The output is also always limited to exactly one vector map (for the same reasons). A future version of this solution will work with raster maps and support progress reports and stored results. Input maps are also currently limited to only one, and it must be a GML map. This is, however, enough to show that by wrapping over an existing GIS, one does not need to re-develop the generalisation operations that do the actual work, but instead, utilise the existing ones that have already been shown to work.

ACKNOWLEDGEMENTS

This research was funded by the Ministry of Agriculture and Forestry, Finland. The authors want to thank Jaakko Kähkönen and Janne Kovanen for their help and support during this project.

BIBLIOGRAPHY

52 North, 52n Web Processing Service, Online:
http://52north.org/maven/project-sites/wps/52n-wps-webapp/, (010109), 2008

Bundala, D. and W. Bergenheim, GRASS GIS: v.generalize, Online:
http://grass.osgeo.org/grass64/manuals/html64_user/v.generalize.html, (010109), 2007.

Burghardt, D., Neun, M. and R. Weibel, Generalisation Services on the Web – Classification and an Initial Prototype Implementation, *Cartography and Geographic Information Science,* **32**(4): 257-268, 2005.

Digiroad, Digiroad – A National Road and Street Database, Online: http://www.digiroad.fi/en_GB/, (0150109), 2009.

Douglas, D. H. and T. K. Peucker, Algorithms for the reduction of the number of points required to represent a line or its caricature, *The Canadian Cartographer,* **10** (2): 112–123, 1973.

Edwardes, A., Burghardt, D. and M. Neun, Experiments in building an Open Generalisation System. In: Mackaness, W. A., Ruas, A. and L. T. Sarjakoski, (eds.), *Generalisation of Geographic Information: Cartographic Modelling and Applications, Series of International Cartographic Association,* Elsevier, pp. 161-175, 2007.

Edwardes, A., Burghardt, D., Bobzien, M., Harrie, L., Lehto, L., Reichenbacher, T., Sester, M., and W. Weibel, Map Generalisation Technology: Addressing the Need for a Common Research Platform, *Proceedings of the 21st International Cartographic Conference (ICC), Cartographic Renaissance,* Durban, South Africa, August 10–16, pp. 170–180, CD-ROM, 2003.

Foerster, T., Burghardt, D., Neun, M., Regnauld, N., Swan, J. and R. Weibel, Towards an Interoperable Web Generalisation Services Framework - Current Work in Progress, 11th ICA Workshop on Generalisation and Multiple Representation, Montpellier 2008, Online: http://ica.ign.fr/montpellier2008/workshop.php, (150109), 2008.

GrassGIS, GRASS Development Team, Geographic Resources Analysis Support System (GRASS) Software. Open Source Geospatial Foundation Project, Online: http://grass.osgeo.org, (010109), 2009.

Harrie L. and M. Johansson, Real-time data generalisation and integration using Java. *Geoforum Perspectiv,* February 2003, pp. 29-34, 2003.

ICA, ICA Commission on Generalisation and Multiple Representation, Workshop – Moscow – August 2007, Online: http://ica.ign.fr/Moscow/program.php, (120309), 2007.

Lehto, L. and L. T. Sarjakoski, Real-time generalization of XML-encoded spatial data for the Web and mobile devices. *International Journal of Geographical Information Science,* 19(8-9): 957-973, 2005.

Lehto, L. and T. Kilpeläinen, Real-time Generalization of Geodata in the WEB. *International Archives of Photogrammetry and Remote Sensing,* Amsterdam, the Netherlands, 16-23 July 2000, XXXIII(B4): 559-566, 2000.

Lehto, L. and T. Sarjakoski, XML in Service Architectures for Mobile Cartographic Applications. In: Meng, L., Zipf, A. and T. Reichenbacher (eds.), *Map-based mobile services - Theories, Methods and Implementations,* Springer Berlin Heidelberg New York, pp. 173-192, 2005.

Lehto, L., Real-Time Content Transformations in a Web Service-Based Delivery Architecture for Geographic Information. Doctoral dissertation, Helsinki University of Technology. *Publications of the Finnish Geodetic Institute,* Kirkkonummi, N:o 138, 2007.

Neteler M. and H. Mitasova, *Open Source GIS: A GRASS GIS Approach.* 3rd edn., Springer, New York, 2008.

OGC, OGC 05-007r7, OpenGIS® Web Processing Service version 1.0.0, Online: http://portal.opengeospatial.org/files/index.php?artifact_id=24151, (160309), 2007.

PHP, PHP: Hypertext Preprocessor, Online: http://php.net/ (010109), 2009.

RecAcro, Online: http://en.wikipedia.org/wiki/Recursive_acronym (010109), 2009.

RFC, RFC 2616, Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. Berners-Lee, T., Hypertext Transfer Protocol – HTTP/1.1, June 1999, Online: http://www.apps.ietf.org/rfc/rfc2616.txt, (160309), 1999.

Sarjakoski, T. and L. T. Sarjakoski. A Real-Time Generalisation and Map Adaptation Approach for Location-Based Services. Chapter 7 in: Mackaness, W. A., Ruas, A. and L. T. Sarjakoski, (eds.), *Generalisation of Geographic Information: Cartographic Modelling and Applications, Series of International Cartographic Association,* Elsevier, pp. 137-159, 2007.

Sarjakoski, T., Sester, M., Sarjakoski, L.T., Harrie, L., Hampe, M., Lehto, L. and T. Koivula, Web generalisation service in GiMoDig - towards a standardised service for real-time generalisation. In: Toppen, F., and M. Painho, (eds.), *Conference proceedings, AGILE 2005, 8th Conference on Geographic Information Science,* Estoril, Portugal, May 26-28, 2005, pp. 509-518, 2005.