The GRASS Server

Radim Blazek*, Luca Nardelli**

* ITC-irst, Via Sommarive 18, Trento, Italy, tel. +39 0461 314 520, e-mail blazek@itc.it
** ITC-irst, Via Sommarive 18, Trento, Italy, tel. +39 0461 314 554, e-mail lunarde@itc.it

1 Introduction

GIS applications based on Internet technology, also known as WebGIS, are nowadays one of the most rapidly growing GIS sectors. Via WebGIS platforms, more and more users using many different kinds of devices, ranging form PC to PDA and mobile phones, are taking advantage geographic information. While most of the current WebGIS applications are oriented toward interactive viewing and querying, tools for GIS analysis are going to be integral part of WebGIS services.

GRASS has a big potential to fulfill the gap of functionality in open source GIS tools, currently mostly used in WebGIS applications, like MapServer [5]. GRASS has a relatively long tradition in WebGIS analysis. Several WebGIS systems using GRASS as the GIS engine for spatial analysis were built in last years. The GRASSLinks [2] is probably the most known example of such systems. Use of GRASS as back-end for WebGIS systems continued in various forms, e.g. [3] and [4].

The web applications using GRASS however were usually based on execution of GRASS modules from CGI scripts. The execution of GRASS modules brings the necessity of temporary mapsets and session management. Other drawbacks are security and performance. The GRASS Server, we present in this paper, makes the use of GRASS in Web applications more robust and secure. The GRASS server is not in any sense a substitution of existing GIS data presentation system like MapServer, it gives however the possibility to add more analytical functions to such systems.

2 Overall design

The GRASS Server is a client-server system, built on top of GRASS, which gives access to GRASS data and functionality for clients via TCP/IP.

The GRASS Server usually does not communicate directly with Web browser or any other end-user's software. The client, from the GRASS Server point of view, is another WebGIS server, actually sending data to users. The client may be for example a PHP application integrating MapServer and GRASS functionality.

The general design of the system is displayed in Figure 1.

The implementation of the system follows these requirements:

1. The server and client must be able to run on two different machines, including different hardware (byte order, etc.) and operating system. It is very important to have the possibility to run the GRASS server on a different machine than the WebGIS server, because of load balancing. Any special requirements for the machines where GRASS server and client are running, would make this more difficult.

2. The response of the server must be fast enough for interactive Web services. This requirement practically excludes any execution of GRASS commands and imposes the necessity to preload data into memory on server side.

3. The system must work without writing to temporary files and without using temporary mapsets.

4. It must be possible to use the system without session/user management.

5. Because of security issues, the system must not execute any external programs.

6. The running server must be able to access data in different databases, locations and mapsets.

7. Access to data must be restrictable on server side.

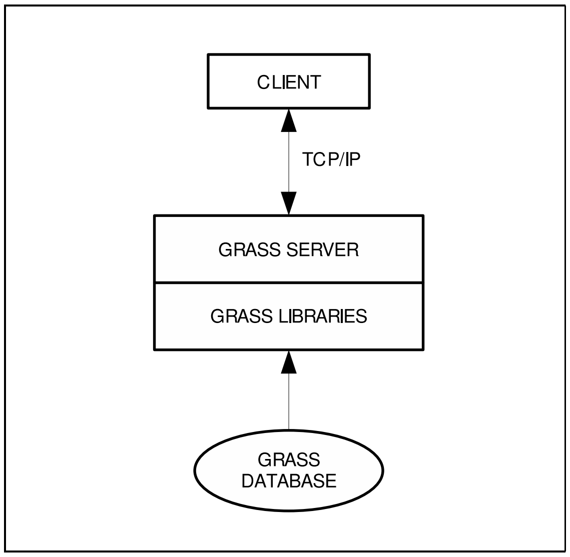

Figure 1: Overall design

3 Server

The existing GRASS library, distributed with GRASS GIS package, is not thread safe, because GRASS was designed as a system of small stand-alone modules and not as a library for multithread programs. Unfortunately this is very serious drawback, especially if Web services are involved. This constraints the design quite a lot and in fact limits the scalability of the server. Because the library is not thread safe, it is impossible to create a new thread for each new request sent by client. The implication is, that one client cannot keep an open connection with the server for too long, because other clients must wait until the previous request is finished. The scalability of the system results limited.

The services provided by the server are ‘read-only’. This means that data can be read from an existing database and analysed, but it is not possible to edit or create data. This could become a problem especially for raster analysis, for which temporary raster layers are often used. On the other hand, raster analysis which require large temporary layers are usually time consuming and do not fit well to the fields where GRASS server is expected to be used, i.e. interactive work with fast responses.

Start of the GRASS server is done in four steps:

1. Reading of configuration file. The configuration file defines some details about the connection and data layers which will be available.
2. Checking availability of data specified in the configuration file.
3. Loading data. Support data for vectors (topology, spatial index, category index) are loaded into the memory and network graph is built when the server is started. This makes the responses much faster than if GRASS modules are executed for each request.
4. Waiting for requests from a client.

4 Client - server communication

To fulfill the requirement of portability the data must be sent in platform independent form. For this purpose the XDR library [6] was chosen. All data sent from client to server and vice versa are encoded by XDR functions. The data are interchanged via standard sockets and TCP/IP protocols.

As we explained above, it is impossible to use threads. To avoid a situation in which one client opens connection and ‘forgets’ to close it, a new connection is opened for each request and the connection is automatically closed by server when the request is finished.

The steps of a request cycle:

1. Client: open a connection to the server

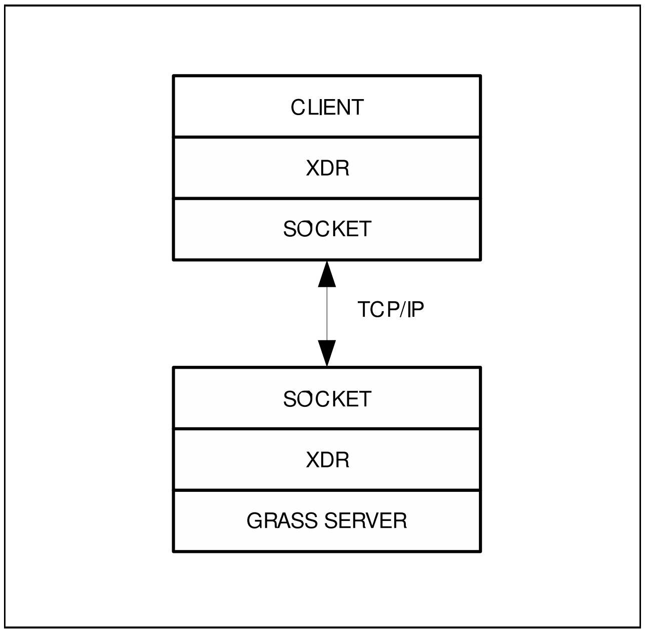

Figure 2: Client - server communication

2. Client: send request
3. Server: receive request
4. Server: check if the data and function requested are available
5. Server: call function in GRASS libraries
6. Server: send response
7. Client: receive response
8. Server: close connection

5 Client

There are no special restrictions for clients. A client can be written in any language with support for sockets and XDR. Currently implemented is client library in C, simple client program in C (mostly for testing purposes) and PHP extension module written in C using the client library in C. Figure 3 shows the schema for clients.

6 Configuration of the server

The configuration is stored in plain text file. Syntax of the configuration file is similar to the syntax of MapServer configuration file. The configuration specifies connection parameters and data layers. A connection parameter is for example number of port on which the server is listening. Only data in the layers specified in the configuration will be available for clients. An example of the configuration file:

```
CONNECTION
    PORT 9300
END

LAYER
    NAME dem
    TYPE RASTER
    GISDBASE /gdata/
```

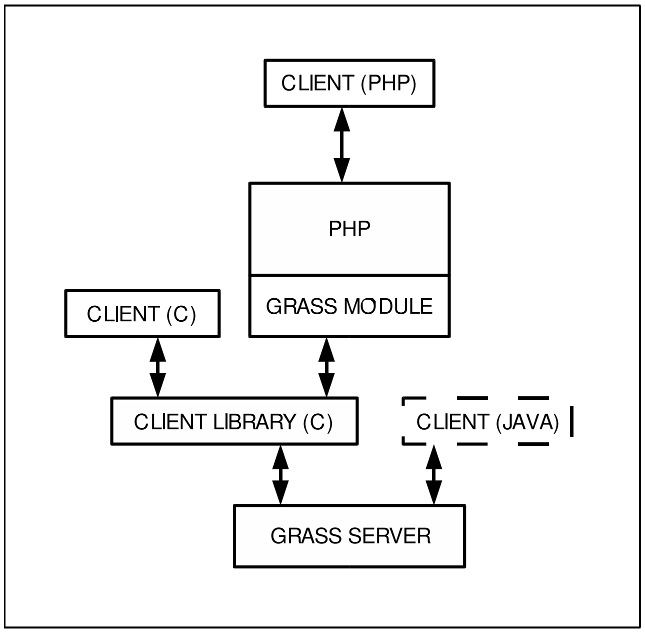

Figure 3: Clients

```
    LOCATION gserver
    MAPSET PERMANENT
    MAP dem100
END

LAYER
    NAME roads
    TYPE VECTOR
    GISDBASE /gdata/
    LOCATION gserver
    MAPSET PERMANENT
    MAP roads
END
```

A server started with this example configuration will listen on port 9300 and it will use two layers. Layer with NAME 'dem' is raster layer, layer with NAME 'roads' is vector layer. Clients access data by the layer NAME, not by the original location in the database. This way it is possible to give access only to certain data.

7 Example PHP client code

A small example in PHP is given, which illustrates how easy it is to use GRASS Server on client side. In this case the client calls the function for shortest path between two points given by coordinates. The shortest path is searched on the network represented by a vector layer. Resulting 'costs' and coordinates are then printed to standard output.

```
<?php
include 'grass.php';

$gs = new grass_server();   // new grass server object
$sp = new grass_points();   // object to store resulting points

$x1 = 17727; $y1 = 11534;   // coordinates of start point
$x2 = 17774; $y2 = 11507;   // coordinates of end point

grass_shortest_path_coor($gs, "roads", $x1, $y1, $x2, $y2, &$costs, &$sp);
```

```
print("Shortest path costs = $costs\n");
for ( $i = 0; $i < $sp->n; $i++ ) {
  $x = $sp->x[$i];
  $y = $sp->y[$i];
  print ( "x = $x y = $y\n");
}
?>
```

In practical applications, the coordinates of the shortest path can be used for example to draw the shortest path on the output map image.

8 Building applications: GRASS Server + MapServer

Because the GRASS Server is not a full featured WebGIS system, to build a Web based applications, other tools, which can render map images, must be used. An example of such tool is MapServer.

To create an application using both GRASS Server and MapServer a programming language supported by both packages must be used. As GRASS Server currently supports only C and PHP clients, PHP is currently the obvious choice in this case. Future versions of the GRASS Server will probably support more languages on the client side.

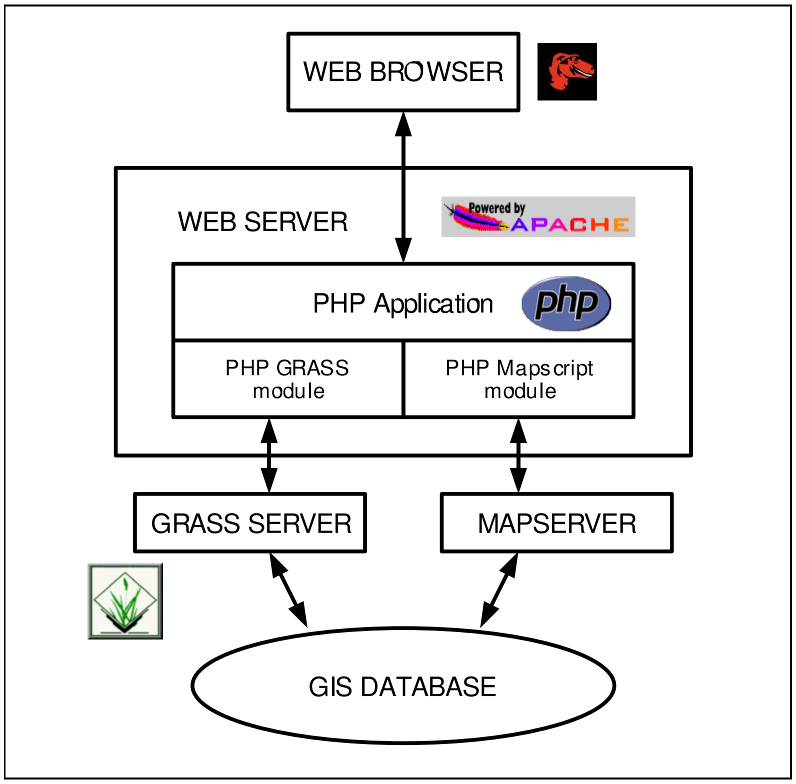

Figure 4: Overall scheme of a WebGIS application

In Figure 4 you can see overall scheme of a WebGIS application using GRASS Server and MapServer.

If we continue with previously given example of shortest path, here is a fragment of PHP code which adds shortest path obtained from GRASS Server to an image rendered by MapServer

```
$spLine = ms_newLineObj();
for ( $i = 0; $i < $sp->n; $i++ ) {
    $spLine->addXY( $sp->x[$i], $sp->y[$i]);
}

$spShape = ms_newShapeObj (MS_SHAPE_LINE);
```

```
$spShape->add($spLine);

$spLayer = $map->getLayerByName('sp');
$spShape->draw($map, $spLayer, $img);
```

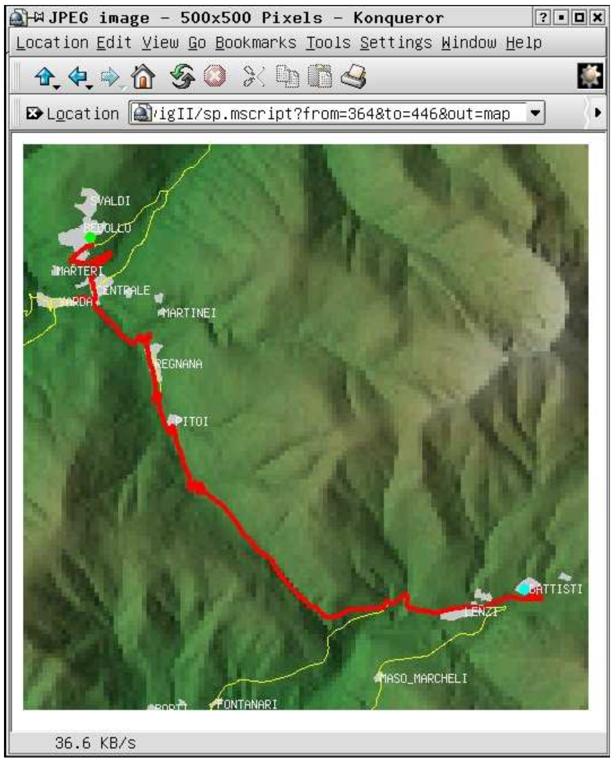
JPEG image - 500x500 Pixels - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location
/igII/sp.mscript?from=364&to=446&out=map
SVALDI
BEDOLLO
MARTERI
CENTRALE
VARDA
MARTINEI
REGNANA
PITOI
BATTISTI
LENZI
MASO_MARCHELI
FONTANARI
36.6 KB/s

Figure 5: Shortest path

Figure 5 shows an example result returned by WebGIS application for request of shortest path between two points, sent by Web browser.

9 Example use in a multimodal browsing system

The GRASS Server was successfully used in an experimental Web based multimodal application developed for the project WILMA [1].

The Multimodal Browsing System (MBS) is capable of handling multi-modal interactions through the synchronization of HTML and Voice-XML documents. HTML documents define a usual interaction based on standard devices, such as: graphic monitor, keyboard, mouse and touch screen, while VoiceXML documents define a corresponding interaction based on voice. The multi-modal browsing system detects events coming from various types of input devices, including a microphone, and provides output according to predefined documents (HTML and VoiceXML). Note that, in this way, the user can freely interact by using the preferred device (e.g. the mouse, for selecting an item from a short list of options, or voice for filling the fields of a form) and the system is able to provide output in a coherent way.

In Figure 6, you can see a photo of the example application running on the PDA. It is possible to select the origin or destination by using the GUI or by voice (e.g. 'I would like to go to Trento'). Once origin and destination have been selected, a new map with the shortest path can be generated (Figure 6). By voice, it is also possible to request zoom operation and/or movement operation (e.g. 'Zoom level 3', 'Go to West', etc).

10 Conclusions

The main problem of the proposed solution is the limited scalability. As described above, the GRASS library currently is not thread safe, and only one request can be processed by the server at the same

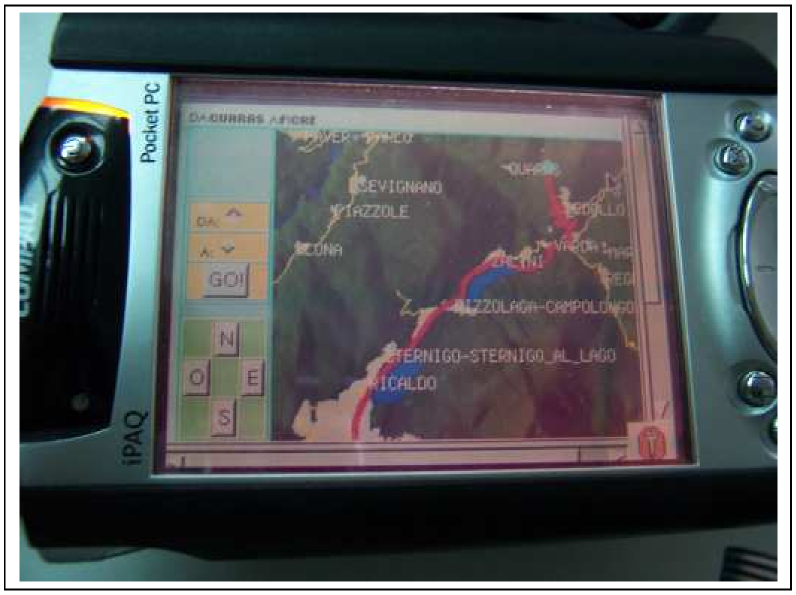

Figure 6: Multimodal application

time.

The system is suitable for analysis with fast response. Attention must be paid to the server load. If necessary, other methods must be used to distribute requests from clients to more running GRASS Servers, either on one or more machines.

The future work will concentrate on adding new functionality and support for more client languages. This should be reached by an automatic generation of the code for both server and client side from a list of function descriptions.

11 Acknowledgement

The work was partly supported by the WILMA project [1].

References

[1] WILMA, URL: http://www.wilmaproject.org/

[2] Huse, S.M., 1995, GRASSLinks: A New Model for Spatial Information Access for Environmental Planning. *Ph.D Thesis, University of California* , URL: http://www.regis.berkeley.edu/sue/phd/

[3] Venkatesh Raghavan, Shinji Masumoto, Phisan. Santitamont, Kiyoshi Honda, 2002, Implementing an online spatial database using the GRASS GIS environment. Proceedings of the *Open Source Free Software GIS - GRASS users conference*

[4] Hess Sigrid, 2002, Grass On The Web. Proceedings of the *Open Source Free Software GIS - GRASS users conference*,

[5] MapServer, URL: http://mapserver.gis.umn.edu/

[6] XDR (External Data Representation Standard), URL: http://rfc.net/rfc1014.html

[7] OpenGIS Consortium, URL: http://opengis.net/