Geoinformatica: a modeling platform built on FOSS

Ari Jolma
Helsinki University of Technology
ari.jolma@tkk.fi

Environmental data analysis and modeling tools need interoperability and capabilities for managing application specific objects. Free and Open Source Software (FOSS) is a successful software development paradigm, which has lead to the availability of a large number of software components that can be exploited. This paper presents and discusses a software stack that integrates three major FOSS components: Perl, GTK, and GDAL/OGR into a geospatial data analysis and modeling platform. All of these three FOSS projects have a significant community support. The capabilities that the platform offers have been enhanced and partly created by code written in the current research project. The platform offers generic programming capabilities; a GUI with analytical, modeling, and visualization support; and it is extendable in several ways. An interesting possibility is to define new geospatial layer types that are specifically targeted for environmental data analysis, modeling, and planning. The platform could be enhanced by one or more FOSS graphics libraries that would enhance its visualization capabilities and support for interactive environmental planning.

1 Introduction

Environmental and civil engineering planning and management rely on efficient tools for analyzing and processing of data. These tools are often linked to simulation, optimization, or other types of models, or the models may be embedded into the tools. The data that is imported into the tools originates from observation networks, is obtained from data providers such as meteorological institutes and land surveys, is input by the users, or is produced by the models. The progress in making the software tools of civil engineers and environmental managers more efficient and more user-friendly has been impressive during the last decades, but the changes in the software industry, new technologies, and new user requirements call for further progress and open up new possibilities.

One of the new requirements is interoperability, which states that it must be possible for multiple users to analyze and work on the same problems and projects with multiple tools with the least amount of friction. Interoperability calls for shared data models and formats or at least good conversion tools. Another new requirement is the need to model and manage application specific objects in the software. "Product models", popular in many other engineering fields, capture the structure of complex products. The machine-understandable descriptions of the components of the products aid the design engineer in her work. One of the new possibilities is the improved hardware, which allows us to work with larger and more datasets and with more powerful programs. Another new possibility is created by the emergence of free and open source software (FOSS), which makes it possible for developers to build large, tightly coupled systems from freely available and free-to-use components.

This paper describes a software system, or rather a software stack that has been built on FOSS. The stack is named "Geoinformatica", and as its name implies, is mostly targeted as a platform for processing, analyzing, and modeling with geospatial data. A general research goal of the Geoinformatica project is to study the architecture of an environmental data analysis and modeling software. Foci of the research include use of FOSS and scripting languages, linking FOSS components, combining a graphical and command line interfaces (GUI and CLI), and extendable architectures. Garlan and Shaw (1994) have defined software architecture as the overall system structure, which includes organizational and control issues and is manifested in various architectural styles. Garlan and Shaw list, e.g., "pipes and filters", "data abstraction and object-orientation", "event-based", "layered", and "repositories" as common architectural styles.

This paper presents and examines Geoinformatica and its enabling technologies and discusses the usability of the approach and its relevance to environmental data analysis and modeling.

2 The Geoinformatica software stack

The software stack of Geoinformatica is depicted in figure 1. The stack can be divided into at least 7 layers, and each layer can contain one or more separate components. The upper layers are usually on a higher abstraction level or closer to the user and application domain. The upper layers also use the services provided by the lower layers. Three main subsystems are important in the architecture: the Perl programming language and shared code base, the GTK GUI toolkit, and the GDAL/OGR geospatial data abstraction and access library.

The Perl is a high-level programming language. The "high-level" implying that the programmer need not care about technicalities such as compilation and memory management. Perl contains advanced built-in features such as a regular expression engine, which is an effective tool for, e.g., writing data input tools, and associative arrays (i.e., *hashes*), which are effective tools for building and using complex data structures. The Perl community has created a large archive of Perl software: the Comprehensive Perl Archive Network (CPAN), which provides extensions to the core Perl.

The GTK toolkit offers a large set of widgets (GUI components) and a platform for writing graphical, event driven software. GTK and GNOME, which is a software desktop that is built on top of GTK, have several closely related FOSS projects. Geoinformatica uses GTK mainly through its Perl interface, which is a separate FOSS project.

The GDAL/OGR geospatial data access library provides a generic API for reading, manipulating, and creating geospatial datasets. The GDAL/OGR library acts also as an interface, for example, to the Proj4 cartographic projection library and to the GEOS library. The GEOS library implements all the standard (as defined by Open Geospatial Consortium) spatial predicate functions and spatial operators. GDAL/OGR is an Open Source Geospatial Foundation (OSGeo) project.

Perl, GTK, and GDAL/OGR are all multi-platform software and they have been ported to several operating systems. The Geoinformatica software stack has been ported to Linux and Windows.

User interface layer (Perl modules Gtk2::Ex::Geo, gnuplot)	
Analysis and modeling script layer (Perl modules)	
Scripting interface layer (Perl bindings modules)	Integration layer (libral)
Data access layer (GDAL/OGR)	GUI tools layer (GTK)
Basic geospatial tools layer (Proj4, GEOS, libral)	Graphics abstraction layer (Cairo)
Data management and serving layer (PostgreSQL, PostGIS, data format libraries)	
System software layer (Perl, Development environment (Windows+MinGW+MSYS or Unix+GNU)	

Figure 1. The software stack of Geoinformatica.

The author's contribution to the Geoinformatica software stack is the Perl bindings to GDAL and OGR (together with a few other developers), the libral C raster algebra library, and some of the Perl modules in the analysis and modeling and in the interface layers. Specifically:

- The GDAL and OGR Perl bindings are based on the generic SWIG-based bindings of GDAL and OGR. SWIG is a FOSS high-level language wrapper and interface generator. The Perl SWIG bindings, as any other language-specific component, consist mainly of a *typemap* file. A typemap file contains details how in some specific cases (as opposed to simple cases that are managed by SWIG default typemaps) data in C structures are mapped to or from data in Perl structures. Other things in the Perl bindings are configurations for building and compiling them and documentation. The bindings are distributed in the main GDAL/OGR package but they are also released as Perl modules in CPAN as modules in Geo::GDAL, Geo::OGR, and Geo::OSR namespaces. The OSR namespace contains the SpatialReference and CoordinateTransformation classes, which also come from the GDAL/OGR library.
- The libral raster algebra library is an independent library written in C. libral can be optionally linked to GDAL, OGR, and GDK Pixbuf (the GTK image system). libral

operates with in-memory integer or floating point rasters and supports all basic algebraic operations and focal, zonal, and global raster methods (Tomlin 1990). libral contains also code for creating a libral raster from a GDAL dataset, for creating a libral vector geometry from an OGR geometry, and for creating a GDK pixbuf and rendering libral rasters and libral vectors on it. libral contains also some specific functions for hydrological terrain analysis.

- The Perl module Geo::Raster is a Perl binding for libral. The Geo::Raster bindings are written in XS, the Perl extension interface language, not in SWIG, which is generic and language-independent. The Perl module Geo::Vector is an additional layer on the top of Geo::OGR modules. Both of these classes (Perl modules usually define classes) are subclasses of Geo::Layer, which is the root class for all geographic layers in Geoinformatica.
- The Perl modules in the Gtk2::Ex::Geo namespace build on the Perl GTK modules, which are Perl bindings for the GTK GUI toolkit. The Gtk2::Ex::Geo::Overlay is a widget class for geographic canvas. An Overlay object manages an image, created by libral from geospatial data, in a scrollable window. The Gtk2::Ex::Geo::Glue is a class, which instantiates several GTK widgets including an Overlay object and manages their interoperation. One of the widgets Glue manages is a simple text input, for which Glue maintains input history. User input commands that originate from the text input widget are pre-processed and fed to a Perl code evaluator (which itself is a Perl function). A Glue object also manages an on-demand link to gnuplot for plotting.

The simplest possible GUI of Geoinformatica with all built-in widgets is shown in figure 2. Although the GUI provides the basic functionality of a desktop GIS as such, the assumption is that for specific data analysis and modeling needs a new GUI with more functionality is developed.

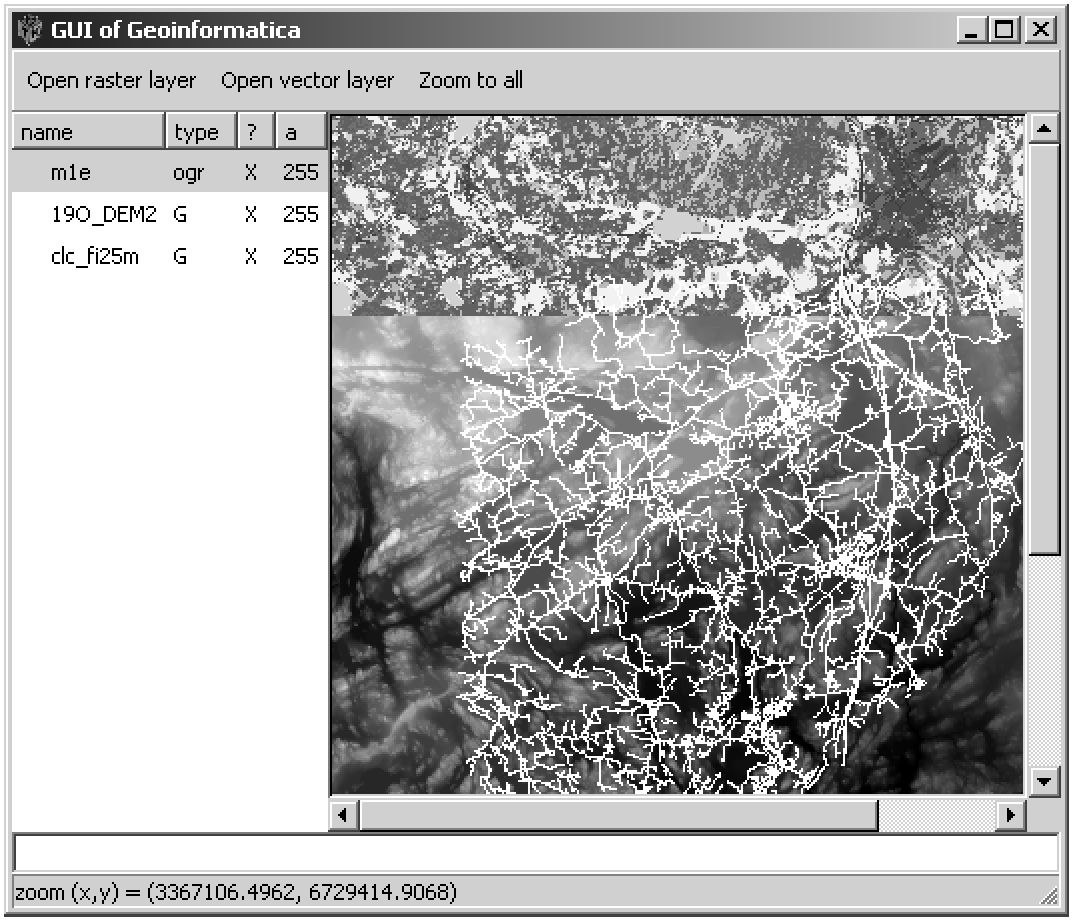

Figure 2. The simple GUI of Geoinformatica. The widgets are from top to down, left to right: a toolbar with three buttons (open raster, open vector, zoom to all), a treeview which displays the stack of open layers, a geographic canvas with scrollbars, the text input field for user commands, the status bar for displaying the interaction mode and information to the user. The picture shows parts of a land cover and DEM rasters and of a road network vector data layer.

3 Data analysis with Geoinformatica

The data analysis functionality of Geoinformatica stem from (i) the analytical capabilities provided by libral and the included Perl modules; (ii) the capabilities for visual analysis of geospatial data provided by the graphical and overlaying functionality of libral and Overlay; and (iii) the general graphical capabilities provided by gnuplot. Besides this functionality, which is readily available, added data analysis functionality can be developed into the stack by linking in other FOSS tools. The author has experience, e.g., from using Graph with Geoinformatica. Graph is a Perl module, which contains a data structure and algorithms for graphs. There are also several Perl modules for statistical analysis, using dates and times, fitting curves, etc. The major FOSS package for statistics is R and a spatial R project exists, but while there is a R-Perl interface project, it has not been tested with Geoinformatica.

The libral provides quite comprehensive analytical capabilites and these can be utilized interactively in the GUI (see below in modeling chapter for more discussion on this) or by writing the data analysis procedure into a Perl program. A limitation is that the libral rasters are completely in-memory and thus very large rasters have to be analysed as tiles. The interaction support through the command line is based on Perl code's capability of executing arbitrary Perl code in runtime. Such capability is common in scripting languages. The command line tool is in the Glue class and it provides an improved support for using the Perl's built-in *eval* function. The pre-processing consists of three steps: (i) the layer names that the user may use are translated into the internal names of the layer objects, (ii) some commands (e.g., *plot*) are expanded into fully qualified method calls ("$self->" is added in front of the method name), and (iii) the new layer, which may spring into existence as a result of the command, is added into the stack of layers.

The visualization capabilities of the GUI are based on the graphics code in libral. The basic functionality is to render all layers into the user-selected bounding box taking into account transparency, coloring, and symbology. The visualization capabilities are thus limited, but at the same time easily extendable in libral by, for example, adding new symbol types and writing respective new C code. A potential remedy to the limitations would be to make use of a generic graphics library – see below for more discussion on this issue. On the other hand, the line drawing algorithm (Bresenham's is used) and other fundamental graphics operatons are re-used in libral from raster methods. The drawing methods in libral can be used to draw into a raster and to extract data from a raster. Libral contains a quite comprehensive coloring system and a similar but more limited symbol system that can be used in rendering rasters and vector data. Supported color palette types are single color, grayscale, rainbow, color table, and color bins. The color table is a hash of integer, color entry pairs and the color bins structure is a hash of bin, color entry pairs. The bin can be either a bin that is defined by two integers or two real numbers. In the GUI there are respective dialogs for interactively defining the coloring and symbol system for a layer. For visual analysis, the color of a cell, a geographic object, or a symbol, or the symbol size can be linked to a value of the cell or an attribute value of the geographic object. There is currently no support for rendering text to the geographic canvas and the support for creating legends is very limited.

The Glue class is able to initiate a link to an external gnuplot program, and instruct it to plot a data file, which it has first created from user-supplied data. Thus, gnuplot can be used to plot for example histograms or longitudinal cross-sections from rasters. gnuplot is in itself a powerful visualization tool. It is also possible to simply print out values from data opened with Geoinformatica. When the command line tool is used, the output goes to the terminal where the GUI was started.

4 Modeling with Geoinformatica

Models can be developed within Geoinformatica or existing models can be linked with it. The support for developing models within Geoinformatica stems from the raster algebra in libral and

especially from the usability that Geo::Raster offers. The standard algebraic equations for Geo::Raster objects are overlaid, using the overlay functionality of Perl, with those provided by libral. Thus it has been made possible to write code like $c = $a + $b, where, if one of $a or $b is a raster object, then $c will be a raster object. The raster methods and overlaid operators are shown in table 1.

Table 1. The main methods in the Geo::Raster class.

Group	Methods	Overlaid operators
Arithmetics	plus, minus, times, over, modulo, power, add, subtract, multiply_by, divide_by, modulus_with, to_power_off	+, -, *, /, %, **, +=, -=, *=, /=, %=, **=
Logical operations	not, and, or	
Comparisons	lt, gt, le, ge, eq, ne, cmp	<, >, <=, >=, ==, !=, <=>
Trigonometric etc. functions	abs, acos, atan, atan2, ceil, cos, cosh, exp, floor, log, log10, sin, sinh, sqrt, tan, tanh	atan2, cos, sin, exp, abs, log, sqrt
Conditional assignments	min, max, if	
Focal methods	count, count_of, min, max, sum, mean, variance	
Zonal methods	count, count_of, min, max, sum, mean, variance	
Global methods	min, max, sum, mean, variance	

The raster algebra methods in libral require that the rasters are of same size and of same geographic location. Thus, the common workflow with Geoinformatica starts with obtaining the datasets; deciding the location to be modeled, the cell size, and the cartographic projection; and preparing the datasets for libral. There is some support for these tasks in the GUI and in libral – clipping and joining rasters and setting their world coordinate systems, but some of these tasks have to be done with, e.g., the command line tools of GDAL/OGR. libral does not have the concept of a map projection.

GDAL supports several data types for rasters, for example 8 bit unsigned integer and 64 bit floating points, but libral supports only single integer and single real data type (the exact data type is a compile time option). Geo::Raster determines in run time whether conversions from integer to real rasters or vice versa are needed and automatically does these conversions without user intervention.

Often data preparation for modeling requires a conversion from vector data to raster or raster to vector. These are supported by Geo::Raster and Geo::Vector, but otherwise there is not very much explicit support for modeling, which employs both raster and vector data at the same time. They can of course be used together and linked via the shared world coordinate system, if their spatial coordinate systems (map projections) are the same.

Both Geo::Raster and Geo::Layer are subclasses of Geo::Layer, which is the root class for all geographic layers in Geoinformatica. The Gtk2::Ex::Geo modules implicitly define the API of a geographic layer by making calls to the layer objects as a result of user actions for displaying the data and opening dialog boxes that list or allow adjusting the layer attributes. Some of these method calls follow a protocol of query for capabilities, show the capabilities for user, use the capabilities as the user wishes. The default behavior of many of the methods is implemented in the Layer class and it is overridden in subclasses.

5 Discussion

Geoinformatica is a platform for geospatial data analysis and modeling. Specific data analysis and modeling tasks and workflows are usually implemented as separate Perl scripts or interactive sessions with the GUI. Similarly, models can be implemented in Geoinformatica as programs that are run from the command line, or as add-ons for the GUI. A typical add-on would be a dialog box, developed with the Glade GUI design tool, and subroutines that are executed as a result of actions on the dialog box.

Our experience shows that a typical data analysis tasks involving geospatial data require a Perl program of roughly 100 lines in Geoinformatica, which usually can be based on generic "read a raster dataset and produce a vector dataset" type programs. If the program seems to become much larger, the programmer is probably overlooking a possibility to use an existing component that could be employed. Using a custom program is usually a step in a data analysis workflow, which otherwise consists of visual analysis and interactive steps with the GUI. The specific benefit of the Perl programming language is that data from arbitrary text files and from formats such as NetCDF, for which there usually is a Perl interface available, can be easily combined with or converted into more common geospatial data.

The mechanism of extending Geoinformatica with deriving new classes from Geo::Layer is particularly interesting. The methods that need to be overridden in a new layer class are all related to the functioning of GUI, thus the extension is mostly related to visualization and interactive use of Geoinformatica. However, possible layer classes, such as *temporal raster* and *catchment management design*, relate also strongly to new capabilities in data analysis and modeling. With classes like these it is possible to imagine following workflows.

- User runs a model on the Geoinformatica platform, which produces a temporal raster. This dataset is then analyzed with methods specific to this data type. The methods include computations and visualizations that can be shown on the map display and/or on

the gluplot as timeseries etc. The temporal raster class itself defines basic graphical user interaction for examining it.

- User zooms into her target location, clips pieces of global datasets, and runs them through standard hydrological methods. She then creates a new layer of the type “catchment management design”, which has an associated ontology. The layer class contains descriptions of management actions and structures that the user interactively designs on a map, constrained by the hydrological system, and being guided by the visualization that the layer can do.

The latter workflow mentions an ontology that is associated with a layer type. Such an ontology is what is referred to as a product model in the introduction. The ontology for a catchment management would contain descriptions of various structures and policies that can be put into use on a catchment. These descriptions would contain things that help the designer to place them onto map and into the hydrological system, and that would also provide input to water quality and nutrient load etc. computations.

Geoinformatica would benefit from a generic graphics library or libraries. The current graphics code in libral works well in basic visualization of geospatial data but lacks for example support for text and other surfaces than raster images. Alternatives could be Cairo, which is used by GTK, which also has a Perl interface, and OpenSceneGraph, which is a 3D toolkit. In theory, the integration of Cairo should not be a big technical problem and it would make it much simpler to write new layer types that have specific visualization needs. The OpenSceneGraph library would be much more difficult to integrate with Geoinformatica.

6 Conclusion

FOSS components can be used to develop platforms and tools for environmental data analysis and modeling. Scripting languages such as Perl, and probably also Python and Ruby, provide efficient tools for integrating various libraries into interoperating stacks as well as for writing data analysis programs. Different data models, data structures, and approaches make it usually necessary to write glue code for integrating libraries, but the results are often worth the effort.

Geoinformatica is a platform, which attempts to combine ease-of-use with readily available powerful capabilities for data analysis and modeling. Capabilities of GDAL and OGR are exploited in such a way that opening of almost any geospatial dataset is easy. Capabilities of Perl are exploited in easy – from a developers point-of-view – and scriptable environment for data analysis and modeling. Capabilities of GTK and Perl-GTK have been exploited in easy development of GUI programs and extensions.

7 References

Garlan, D. and Shaw, M. 1994. An introduction to software architecture. Advances in Software Engineering and Knowledge Engineering, Volume I, edited by V.Ambriola and G.Tortora, World Scientific Publishing Company, New Jersey, 1993.

Tomlin, C.D. 1990. Geographic Information Systems and Cartographic Modelling. Prentice-Hall 1990.