Extending GIS with Object-Orientation for Environmental Planning and Modeling

Jolma, A.

Helsinki University of Technology, Finland
Email: ari.jolma@tkk.fi

Keywords: *Geospatial software, dynamic modeling, object-orientation, ontology*

EXTENDED ABSTRACT

Using and linking geographic information systems (GIS) efficiently with environmental modeling remains a challenge. Several solutions have been proposed for the impedance mismatch problem between the data oriented domain of GIS and the modeling oriented domain of environmental planning and management. Concepts that are important for dynamic modeling, i.e., time and interaction of different type of objects, are difficult to represent in standard GIS. However, geoprocessing operations for raster and vector layers are well-defined and standardized. In this paper we demonstrate with a free and open source software how object-orientation can be used to define new spatial layer classes that are close to environmental planning and modeling.

Computer programming and development of information systems advances through championing increasingly complex subjects in machines. Object-oriented programming (OOP, also OO analysis, OO data modeling, etc) came into mainstream use in early 90s but the paradigm was developed in preceding decades. One of the strong benefits of OOP is the possibility to implement abstract ideas or generalizations of facts, i.e., design patterns (Gamma et al 1994) or predicate logic, in program code more explicitly that with earlier programming paradigms. Ontological engineering uses very similar concepts to OOP but it strives to being more than of just immediate practical benefit. Ontologies are increasingly also a subject of study in geographic information science. A major impetus for this research is the availability of huge collections of geospatial data that need to be integrated in a meaningful way (Fonseca et al. 2002).

In this paper we describe how object-orientation is applied in software called Geoinformatica to build geospatial software for environmental modeling. A layer is a fundamental paradigm in geospatial software. At the core of Geoinformatica there is a definition of an abstract geospatial layer. The abstract layer class defines an object that can be visualized with other layers, used as a component in a simulation model, and manipulated within an interactive programming environment.

In Geoinformatica the two traditional layer classes, raster layer and vector layer, derive from the root class and add respective computational capabilities. The raster layer class combines the visual methods with map algebra and the vector layer class combines the visual methods with the simple features analytical methods for geometric objects. New layer classes can be defined easily by inheriting from the root layer class and from classes that provide computational capabilities.

For this study three new layer classes were developed: (i) a class, which maintains a list of geospatial features in RAM and does not assume that they share a geometry shape nor attributes, (ii) a class, which acts as an interface to a raster time series, and (iii) a simple multi-agent simulator, which consists of a raster object and of a feature list class. The feature list class utilizes the generic geospatial feature class from the Perl bindings to the OGR library, which provides access to several vector data formats. The raster time series class was developed mainly for visualization purposes. It expects a simple text file in its constructor from which it reads the filenames of the raster files and associated dates. The other code of that class defines a simple dialog box and associated code for opening and showing or animating the requested raster files. The simple multi-agent simulator layer class code consists mostly of callback functions that respond to user actions and implement the model dynamics. A multi-agent simulator object contains the agents as geospatial features object and the world as a raster object.

The study demonstrates that it is possible to develop a framework for spatial environmental modeling with current free and open source software: high-level programming language, graphical user interface toolkit, and geospatial data access software. Using object-orientation the geospatial layer paradigm can be extended for environmental modeling and software.

1. INTRODUCTION

Using and linking geographic information systems (GIS) or any geospatial software efficiently with environmental modeling remains a challenge. Several solutions have been proposed for the impedance mismatch problem between the data management oriented domain of GIS and the modeling oriented domain of environmental planning and management. Proposed solutions are based on methodology, data processing, and/or technology. The most important methodology-based solution is map algebra (Tomlin 1990, DeMers 2002, Pullar 2003), which has been implemented in most raster GISs in some way and offers a spatial modeling language for building dynamic simulation models. The simple features data model and spatial access and analysis methodology (Herring 2006) defines a somewhat similar spatial modeling language for geometric objects, but there are few reports of embedding it into simulation models. Miller et at (2005) discuss dynamic simulation models and GIS functionality and present also an approach for integrating them using a formalism that is based on a graph of datasets (that are not limited to raster layers) and tools. Solving environmental models expressed as differential equations in 2D finite element meshes is a standard methodology but it commonly uses other specific software (e.g., CAD) and is linked to GIS via data exchange. Solutions based on data processing use the GIS as a pre-processing tool to create datasets for special purpose environmental models or management tools and/or as a post-processing visualization tool.

Models and tools may be linked to or integrated with GIS using different technological solutions. The technological solutions and especially the impact of personal computers and workstations that allow interactive graphics and user-centric computing have been studied since 1980's (Loucks and Fedra 1987, Fedra 1993, Lam and Pupp 1994). From information technology point-of-view the GIS is a set of functionalities that are embedded in an environmental information system aimed to support planning, management, decision making, modeling, analysis, etc. Different technologies (data formats, client-server communication, remote procedure calls, etc.) allow embedding and linking GIS functionalities more or less tightly with other functionalities.

Computer programming and development of information systems advances through championing increasingly complex subjects in machines. Progress is made possible because of well-defined and usable foundations. Object-oriented programming (OOP, also OO analysis, OO data modeling, etc) came into mainstream use in early 90s but the paradigm was developed in preceding decades. One of the strong benefits (but also the thing that makes it hard) of OOP is the possibility to implement abstract ideas or generalizations of facts, i.e., design patterns (Gamma et al 1994) or predicate logic, in program code more explicitly that with earlier programming paradigms. The simple features geometry model mentioned above is a good example of the current state-of-the art of utilizing OO in GIS. In a geospatial environmental application one would define a class to represent a class of environmental objects, e.g., forest stand, and let it inherit the geometric properties and, more importantly, usability in certain spatial operations, from an appropriate geometric class (probably a polygon in this case). OO data models for environmental domain exist, perhaps most notably the Arc Hydrology Data Model (Davis *et al* 2000).

The next step forward from OOP in championing increasingly complex subjects with machines is through ontologies. Ontology is a philosophical subject that studies "the nature of knowable things" (Wikipedia.org), but also an engineering subject that develops ontologies, i.e., data models that are used in various areas to represent knowledge about some domain. Ontological engineering uses very similar concepts to the ones used in OOP but it strives to being more than of just immediate practical benefit. Ontology tools, such as the CycL ontology language (Cycorp 2002), also attempt to maintain a link to predicate logic. Ontologies are increasingly also a subject of study in geographic information science. A major impetus for this research is the availability of huge collections of geospatial data that need to be integrated in a meaningful way (Fonseca et al. 2002). In artificial intelligence propositional and predicate logic have been used to develop reasoning systems (Russell and Norvig 2003), where the model refers to a system of causal rules written in predicate (first-order) logic.

The goal of this study is to define and implement an OO interface to a geospatial dataset in a GIS that can be used as a foundation in environmental modeling and planning tasks. The expected result is a software framework that has the state-of-the-art GIS functionality but offers also a simple platform for developing software and applications for the environmental domain. As materials, this study uses free and open source (FOSS) software. The current state-of-the-art of geospatial FOSS and its applicability for environmental modeling and management has been reviewed by Jolma *et al.* (2006). The software stack, which offers the OO interface is referred to as Geoinformatica (Jolma 2007).

2. THE GEOINFORMATICA SOFTWARE STACK

The software stack of Geoinformatica consists of several software layers that usually at least partially depend on lower layers. The following is a short description of each layer, starting from the most basic ones:

1. The system software layer, which contains mainly the core Perl programming language, the operating system (the free alternative is Linux), and a software development platform. A major free software development platform alternative is the GNU system that consists of compilers and other tools. An important part of the development platform is the Glade user interface designer tool, that can be used to develop GUIS and store them as XML files.

2. The data management and serving layer, which contains mainly a database management system (PostgreSQL and PostGIS is the preferred free solution for geospatial data), libraries and Perl modules and programs for accessing data in various formats.

3. The basic geospatial tools layer, which contains C and C++ libraries such as Proj4 (coordinate system conversions), GEOS (simple feature methods), and libral (raster algebra methods).

4. The graphics abstraction and rendering layer, which currently contains the Cairo 2D graphics library, the GDK graphics for X, specific support code that is built on top of these tools, and the data and function plotting utility gnuplot.

5. The GUI toolkit layer that is provided by the GTK+ toolkit and Gtk2, its Perl interface.

6. The geospatial data access and abstraction layer, which contains the GDAL and OGR libraries for raster and vector data respectively. On this layer there is also the Geo::OGC::Geometry Perl module that currently provides a storage for geospatial data that adheres to the Open Source Geospatial Foundation (OGC) simple feature geometry model.

7. The high-level programming (scripting) layer for geospatial data, that is provided by the Perl modules Geo::GDAL, Geo::Vector, and Geo::Raster.

8. The abstract GIS layer, a geocanvas, and a layer stack, which are the main foci of this paper.

9. The standard raster and vector geospatial data layer implementations of the abstract GIS layer that build on Geo::Vector, and Geo::Raster.

The last three layers (7-9) are almost pure Perl modules. Perl is a high-level programming language (HLL). 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. Perl supports OO but it does not expect everything to be an object. Efficient Perl bindings to C and C++ libraries can be built with Perl’s xs system or using the HLL agnostic SWIG (http://www.swig.org).

The GTK toolkit offers a large set of widgets (GUI components) and a platform for writing interactive, 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 uses internally the Proj4 cartographic projection library and 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. The GDAL/OGR library has Perl SWIG bindings, which define 19 classes in the Geo::GDAL, Geo::OGR,and Geo::OSR namespaces.

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.

3. AN ABSTRACT GIS LAYER

The 8th (previous chapter) layer in the Geoinformatica stack defines an abstract GIS layer. The formal name of the class is Gtk2::Ex::Geo::Layer.

A layer is a fundamental paradigm in practical geospatial software. Geospatial datasets are usually either continuous fields or collections of objects

that belong to same feature class and share their geometric shapes and attribute schemas. When creating a map or a geospatial visualization the geospatial datasets are rendered on top of each other in a certain order and using specific visual parameters to produce an aesthetically pleasing and/or informative visualization. Many analytical GIS tools consider overlays of geospatial objects or datasets. An overlay is similar to rendering datasets on top of each other, but it is done computationally, producing new geospatial datasets and not just visual images.

The abstract layer class defines mainly an object that is designed to be in a GUI. The class methods (the methods that are shared by each object of the class) return, e.g., dialog box classes (each layer object normally has its own dialog box instance of each class), application context commands, and object context menu items with associated callback functions. The GUI defines how the application context commands and object context menu items are presented to the user. The object methods (the methods that affect the individual object they are applied to) are mostly about the visual features and properties of the layers. Currently the root layer class defines four main visual properties:

1. Visibility and transparency: a layer can be invisible or its transparency can be set in the scale of 0 (invisible) to 255 (opaque).

2. Colors: a layer has a palette type, which is one of the palette types it supports. The list of supported palette types is a class method. The actual coloring of the layer can be based on the value of the selected attribute of each geometric object. The root layer class defines a coloring dialog box, which supports common palette types grayscale, rainbow, color table and color bins.

3. Symbols: a layer has a symbol type, which is one of the symbol types it supports. If a symbol type is set for a layer object, all its objects are rendered using that symbol in a location specified by the location of the object (currently centroid). The list of supported symbol types is a class method. Currently the symbol size is the only property (besides color of course) that can be linked to an attribute value.

4. Labels: an attribute value can be rendered next (the placement can be defined) to a location of the object (currently centroid) using a definable font and color.

In addition to the visual features the root layer class API uses the concept of a geospatial feature and of a schema. A layer is assumed to consist of features, a subset of which may be selected. The schema is defined because having visual features that may be linked to attributes require it.

This set of properties lacks some common visualization tools: line and fill styles for example. It is also notable that the properties adhere to, as is common in GIS, to the layer as whole.

A geocanvas object contains a list of geospatial dataset layers. When a geocanvas object gets a request for rendering, it calls the render method of each dataset layer in order. The parameters of the render method specify the size and coordinates of the canvas and provide a graphics context.

4. STANDARD LAYER CLASSES

The standard GIS layer classes are a geospatial raster class and a geospatial vector dataset class. These classes are implemented by joining the functionality of a data access / modeling class with the functionality of a visualization / GUI tool class.

Gtk2::Ex::Geo::Raster is a class that inherits both (Perl supports multiple inheritance) Geo::Raster and Gtk2::Ex::Geo::Layer. Geo::Raster is mostly a Perl interface to the libral raster algebra library but it can also be used to open any raster dataset with the aid of the general GDAL raster access library and retrieve data from it into the libral raster for manipulation. Thus Geo::Raster brings into the class powerful data access and raster modeling support. The class binds the modeling functionality with the visualization capability in a class that is a geospatial layer and usable in a GUI. Raster commands “open”, “save”, “save all”, “clip”, and “vectorize”, and dialog box classes for setting the properties of the raster layer object are accessible from a GUI.

Gtk2::Ex::Geo::Vector inherits Geo::Vector and Gtk2::Ex::Geo::Layer. Geo::Vector is a class that represents a vector layer that is accessed with the general OGR vector library. OGR supports most of the common geospatial vector formats and provides a way to use the data with the GEOS simple feature geometry method library. Thus Geo::Vector, similarly as Geo::Raster, brings into the class powerful data access and vector computation support. Vector commands “open”, “features”, “vertices”, “clip”, and “rasterize”, and seven dialog box classes for setting the properties of the vector layer object, are accessible from a GUI.

5. NEW LAYER CLASSES

The main focus of this study is to examine how GIS can be extended as an modeling, simulation,

and planning tool through OO. The two previous chapters have described how state-of-the-art, with some limitations, desktop GIS was built from FOSS using OOP and a HLL. In this chapter we examine how this foundation can be easily extended using OOP.

New geospatial layer classes can be defined easily by inheriting from the root layer class or from the standard raster or vector classes. With multiple inheritance such class can be equipped with computational capabilities using suitable other classes. Examples of functionality that could be integrated into the system this way are the Graph module (class) (Orwant et al 1999), and possibly even R spatial using the R-Perl interface (http://www.omegahat.org/RSPerl/).

Listing 1 shows a very simple new class declaration. The declaration consists or two header rows (1-2) and five methods of which the first (lines 3-14) is a class method, the second (lines 15-18) is the constructor, and the three last ones ("world", "render", and "menu_items") are object methods – although the third method (lines 19-21) is functionally a class method and defines static world for the objects of this class. The render method (lines 22-24), a method used by the geocanvas, does in the example absolutely nothing else than retrieving its parameters, i.e., nothing is rendered on the geocanvas. The fifth method provides the object context menu_items for the GUI, which first retrieves the default methods defined by the root class and then adds a menu item, whose callback function (lines 30-32) does also nothing.

Listing 1. A very simple new layer class declaration.

```
1 package MyLayerClass;
2 our @ISA = qw(Gtk2::Ex::Geo::Layer);
3 sub registration {
4    my $commands = {
5      open => {
6          text => 'Test',
7          tip => 'A test command.',
8          sub => sub {
9             my(undef, $gui) = @_;
10         }
11      }
12   };
13   return {commands => $commands};
14 }
15 sub new {
16   my($class, %params) = @_;
17   return Gtk2::Ex::Geo::Layer::
       new($class, %params);
18 }
19 sub world {
20   return (0, 0, 100, 100);
21 }
22 sub render {
23   my($self, $pb, $cr, $overlay,
       $viewport) = @_;
24 }
25 sub menu_items {
26   my($self, $items) = @_;
27   $items = $self
       ->SUPER::menu_items($items);
28   $items->{'Test'} =
29     {
30        sub => sub {
31         my($self, $gui) = @{$_[1]};
32        }
33     };
34   return $items;
35 }
```

For this study three new layer classes were developed: (i) a feature list class, which maintains a list of geospatial features in memory and does not assume that they share a geometric shape nor schema, (ii) a class, which acts as an interface to a raster time series, and (iii) a simple multi-agent simulator, which consists of a geospatial raster class and of a feature list class.

The feature list class utilizes the Geo::OGR::Feature class from the Perl bindings of OGR. It is rather similar to Geo::Vector and Gtk2::Ex::Geo::Vector but defines its own dialog class for browsing the features and overrides some data manipulation methods.

The raster time series class was developed mainly for visualization purposes – although one can easily think of many interesting and useful methods that such a class could have. The raster time series class expects a simple text file in its constructor from which it reads the filenames of the raster files and associated dates. The other code of that class defines a simple dialog box and associated code for opening and showing or animating the requested raster files.

The simple multi-agent simulator layer class code consists mostly of callback functions that respond to the changes in the dialog box (fig. 1) and implement the model dynamics. A multi-agent simulator object contains the agents as geospatial features and the world as a raster.

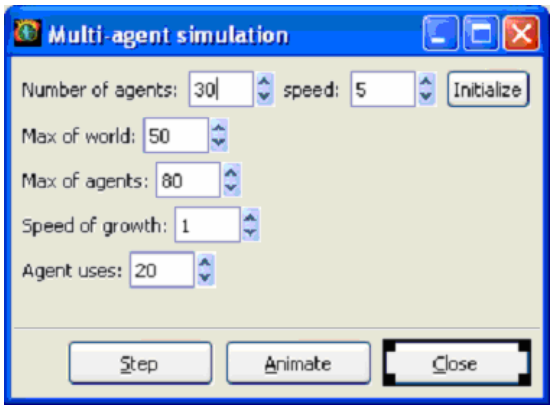

Figure 1. The dialog box for controlling the multi agent simulation.

6. DISCUSSION

The state-of-the-art geospatial FOSS provides tools with which it is quite easy to implement a framework for spatial environmental modeling and software. There are unquestionably also proprietary alternatives to FOSS but the main benefit of FOSS in this case was the open access to all parts of the software and free modifiability. The downside of FOSS in this case was that the standard OO GIS classes did not exist, at least for the chosen HLL (Perl), and had to be implemented first.

Almost any part of the Geoinformatica platform is not exceedingly powerful or novel as such, but as the parts are put together and attention is paid to the OO API of the geospatial dataset in the application, the benefits become obvious. The possibility to start thinking generically about simulation models, into which, once implemented, one can readily import real-world data is intriguing. To be able to connect geospatial visualizations with modeling and use of models is also interesting. The concept of a geospatial data layer gets new power as it is considered from the point-of-view of domain specialist: environmental plan could be a layer, and models should be linked to it as methods of the plan layer class.

Interestingly, the environmental modelers sometimes feel that GIS technology is controlled by people who have different aims than them (van Deursen et al 2000). In the light of this study there really is a gap between the GIS technology and GIS research, and environmental modelers' needs. The GIS research focuses on data, data modeling, and also the research on ontologies seems to be driven by this need. At the same time an environmental modeler is mostly interested in (real world) entities in environmental domain, which certainly have a geospatial existence and thus can benefit from geospatial methods, but most of all have important aspatial attributes and links that need to be modeled as well. Thus, from the environmental modeler's point-of-view, object-orientation and engineering ontologies are welcome tools, but they need to be used to overcome the artificial barriers created by technological development choices.

One conclusion from this study is that high-level languages, Perl in this case, are extremely suitable for rapid development and for developing flexible, almost semi-intelligent software. The downside is, if it can be called as such, that using such a language is also a personal learning experience that never ends. Object-oriented GUI code with callbacks can be powerful but it is also complex. The Geoinformatica software stack, especially the Perl classes developed by the author, strives to be a tool for environmental geocomputations and geospatial applications, but it also is a research platform and these goals are sometimes difficult to combine. This paper presents a snapshot of the development and there are several details in the architecture that still need more thought. For example, what really should be in the root layer class? Should it contain the concepts feature and schema, especially when it is easy to think about layers that contain features with different schemas (as we have done in this paper)?

In the introduction I wrote: "computer programming and development of information systems advances through championing increasingly complex subjects in machines." Certainly when there is more experience gained from applications and modeling projects, new abstract classes can be devised that capture the essential elements of these. It also seems clear that it is not possible to think about these beforehand by theoretical considerations, i.e., the research on information system and software requires practical work with them.

7. REFERENCES

Cycorp 2002. The Syntax of CycL. http://www.cyc.com/cycdoc/ref/cycl-syntax.html (accessed July 26 2007)

Davis, K.M., Whiteaker, T.L., and Maidment, D.R. 2000. Definition of the Arc Hydrology Data Model. Paper Presented at the GIS in Water Resources Conference, University of Texas at Austin 23-25 February, 2000. (http://www.crwr.utexas.edu/giswr/resources/library/archydro.pdf, accessed 24.7.2007)

DeMers, M. 2002. GIS modeling in raster. John Wiley et sons.

van Deursen, W., Wesseling, C., and Karssenberg, D. 2000 How do we gain control over GIS technology? 4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs. Banff, Alberta, Canada, September 2 - 8, 2000.

Fedra, K. 1993. Models, GIS, and expert systems: Integrated water resources models. In: Application of Geographic Information Systems in Hydrology and Water Resources Management. K. Kovar, H.P. Nachtnebel (eds) 297-308. IAHS publication 211.

Fonseca, F.T., Egenhofer, M.J., Agouris, P., and Câmara, G. 2002. Using Ontologies for Integrated Geographic Information Systems. Transactions in GIS 6(3).

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Addison Wesley.

Herring, J.R. 2006 (ed). OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture. Open Geospatial Consortium Inc. OGC 06-103r3

Jolma, A., D.P. Ames, N. Horning, M. Neteler, A. Racicot, and T. Sutton. Free and Open Source geospatial tools for environmental modeling and management. In A. Voinov, editor, Proc. iEMSs 2006, Session W13, July 9-13, 2006, Burlington, Vermont, USA, 2006.

Jolma, A. Geoinformatica: a modeling platform built on FOSS. In: ISESS 2007, Prague, Czech Republic, May 22 - 25, 2007.

Lam, D. and Pupp, C. 1996. Integration of GIS, Expert Systems, and modeling for State-of-Environment Reporting. In: M. Goodchild et al (eds) GIS and Environmental modeling: progress and research issues. GIS World Books.

Loucks, P. and Fedra, K. 1987. Impact of changing computer technology on hydrology and water resource modeling. Review of Geophysics 25(2).

Miller, I., Knopf, S., and Kossik, R. 2005. Linking General-Purpose Dynamic Simulation Models with GIS. In: Maguire et al (eds) GIS, Spatial Analysis and Modeling. ESRI Press.

Orwant, J., Hietaniemi, J., and Macdonald J. (1999). Mastering algorithms with Perl. O'Reilly and Associates.

Pullar, D. 2000. Embedding map algebra into a simulator for environmental modeling. 4th International Conference on Integrating GIS and Environmental Modeling (GIS/EM4): Problems, Prospects and Research Needs. Banff, Alberta, Canada, September 2 - 8, 2000.

Russell, S.J. and Norvig, P. 2003. Artificial Intelligence: A Modern Approach 2nd Edition. Prentice Hall.

Tomlin, C.D. 1990. Geographic Information Systems and Cartographic Modeling. Englewood Cliffs: Prentice Hall.