This is a chapter on the forthcoming book
G. Brent Hall (ed), **"Open Source Approaches to Spatial Data Handling"**
Berlin, Springer-Verlag, 2008

TerraLib: An Open Source GIS Library for Large-scale Environmental and Socio-economic Applications

Gilberto Câmara[1], Lúbia Vinhas[1], Karine Reis Ferreira[1], Gilberto Ribeiro de Queiroz[1], Ricardo Cartaxo Modesto de Souza[1], Antônio Miguel Vieira Monteiro[1], Marcelo Tílio de Carvalho[2], Marco Antonio Casanova[2], Ubirajara Moura de Freitas[3]

Abstract

This work describes TerraLib, an open-source GIS software library. The design goal for TerraLib is to support large-scale applications using socio-economic and environmental data. TerraLib supports coding of geographical applications using spatial databases, and stores data in different DBMS including MySQL and PostgreSQL. Its vector data model is upwards compliant with Open Geospatial Consortium (OGC) standards. It handles spatio-temporal data types (events, moving objects, cell spaces, modifiable objects) and allows spatial, temporal and attribute queries on the database. TerraLib supports dynamic modeling in generalized cell spaces, has a direct runtime link with the R programming language for statistical analysis, and handles large image data sets. The library is developed in C++ and has programming interfaces in Java and Visual Basic. Using TerraLib, the Brazilian National Institute for Space Research (INPE) developed the TerraView open source GIS, which provides functions for data conversion, display, exploratory spatial data analysis and spatial and non-spatial queries. Another noteworthy application is TerraAmazon, Brazil's national database for monitoring deforestation in the Amazon rainforest, which manages more than 2 million complex polygons and 60 gigabytes of remote sensing images.

[1]National Institute for Space Research (INPE), Brazil,
[2]Catholic University of Rio de Janeiro (PUC-RIO), Brazil,
[3]Space Research and Applications Foundation (FUNCATE), Brazil

1. INTRODUCTION

Recent advances in spatial databases have changed GIS software development. Spatially enabled database management systems (DBMS) such as PostgreSQL empower a transition from monolithic GIS with hundreds of functions to a generation of spatial information applications tailored to specific user needs. These capacities have been a major boom for the free and open source geospatial (FOSS4G) community, many members of which are using the new generation of databases to build unique and innovative applications.

One of the expected impacts of open source software (OSS) is its benefits for developing nations. As Weber (2004) points out, combining OSS with the technical workforce available in developing countries can enable technology transfer. He states, "*Of course information technology and open source in particular is not a silver bullet for long–standing development issues; nothing is. But the transformative potential of computing does create new opportunities to make progress on development problems that have been intransigent*" (Weber 2004 p. 254).

GIS is a key technology for developing nations, in areas such as environmental protection, urban management, agricultural production, deforestation mapping, public health assessment, crime-fighting, and socio-economic measurements. The demands of these applications go well beyond the current specifications of the Open Geospatial Consortium (OGC). Large-scale environmental and socio-economic applications compel FOSS4G to include significant spatial analysis capacities (Goodchild 2003). To meet the needs of end-users, FOSS4G should incorporate research advances in areas such as spatio-temporal data models (Erwig and Schneider 2002; Hornsby and Egenhofer 2000), geographical ontologies (Fonseca, Egenhofer, Agouris, and Camara 2002), spatial statistics and spatial econometrics (Anselin 1999), cellular automata (Couclelis 1997), and environmental modeling (Burrough 1998). These results have largely been outside the reach of the GIS user community, due to a general lack of widely available tools that support them. Incorporation of new techniques into GIS applications is necessary for the user community to extract the full potential of spatial databases.

With this motivation, the authors of this chapter are developing TerraLib, an open source GIS software library that extends object-relational DBMS technology to support spatio-temporal models, spatial analysis, spatial data

mining and image databases. The design goal for TerraLib is to support large-scale applications using cadastral, socio-economic and environmental data. This goal was a mandate of the main organization that supports TerraLib, the Brazilian National Institute for Space Research (INPE). INPE is Brazil's main institution for space science and technology, whose mission includes building satellites, developing environmental applications, and producing weather and climate forecasts. Since 1984, INPE has had a research and development division for GIS to support its actions in earth observation and to promote GIS and remote sensing technology in Brazil. The two other main project partners are the Computer Graphics Group (TecGraf) of the Catholic University of Rio de Janeiro (PUC-RIO) and FUNCATE, a non-profit foundation that develops GIS applications using OSS. All organizations share the same design goals. Thus, TerraLib is a product with long-term support and whose programmers have stable jobs. This chapter describes the TerraLib library, explains the main design decisions and points out how the library incorporates research results from GIScience.

2. CHALLENGES FOR INNOVATION IN FOSS4G

The OGC specifications provide a sound basis for developing FOSS4G. However, many emerging applications need tools which go beyond these specifications. Thus, one of the lines of growth in FOSS4G is to provide new tools for application developers. However, there are pitfalls. Building innovation in open source GIS is a threefold challenge. Given the design goals for the product, the first step requires selecting, from the large body of GIScience literature, those advances that are relevant to the product's objectives. Then, these advances need to be implemented in industrial-strength code. The final hurdle is documenting these features for sharing them with the broader FOSS4G development community.

A basic design objective for TerraLib is to support innovative applications for helping people and protecting the environment. Thus, we evaluated current GIS research and selected ideas and proposals that were relevant to our goals. This led to concentration in the following three areas:

(a) *Spatial Statistics*: since Anselin's pioneering work on spatial analysis (Anselin 1989), there appeared relevant results on spatial statistics and spatial data mining (Anselin 1995; Fotheringham, Brunsdon, and Charlton 2002; Openshaw and Alvanides 2001; Martin 2003). The main focus of these contributions is to

improve our ability to extract information for socio-economic data. This is relevant for public policy applications of GIS.

(b) *Spatio-temporal Models*: there are two broad categories of spatio-temporal objects. The first concerns *moving objects*. Moving objects arise, for example, when there is information about spatial and temporal positions of planes, storms or cars. The widespread availability of location-based applications motivates the field of moving object databases (Güting and Schneider 2005). There is a large research area in algorithms and query methods for moving objects (Sistla, Wolfson, Chamberlain, and Dao 1997). The second type concerns *evolving objects* that do not move, but whose geometry, topology and properties change. They arise when we consider the changes that occur in cadastral GIS or in land cover patterns (Medak 2001). Evolving objects are important for environmental models, which depict the temporal evolution of a pattern in a landscape. Examples of environmental models include land change models, epidemiological studies, population flows, and ecological mapping (Burrough 1998; Veldkamp and Fresco 1996).

(c) *Remote Sensing, Image Processing, and Image Databases*: remote sensing satellites are the most significant source of new data about our planet, and remote sensing image databases are the fastest growing archives of spatial information. New high resolution optical sensors and polarimetric radars have improved application areas such as environmental monitoring and urban management. There are important recent results in object-oriented segmentation and classification and in remote sensing data mining (Blaschke and Hay 2001; Navulur 2006; Aksoy, Koperski, Tusk, and Marchisio 2004). It is also important to include support for raster data handling in open-source DBMS, following research results (Chang, Yan, Dimitroff, and Arndt 1988; DeWitt, Kabra, Luo, Patel, and Yu 1994).

To translate these ideas to industrial-strength code, the developers of TerraLib first built up research results (Pedrosa, Câmara, Fonseca, and Souza 2002; Almeida, Batty, Monteiro, Câmara, Soares-Filho, Cerqueira, and Pennachin 2003; Vinhas, Souza, and Câmara 2003; Carneiro, Aguiar, Escada, Câmara, and Monteiro 2004; Ferreira, Vinhas, Queiroz, Câmara, and Souza 2005; Silva, Câmara, Souza, Valeriano, and Escada 2005; Assunção, Neves, Câmara, and Freitas 2006; Feitosa, Câmara, Monteiro, Koschitzki and Silva 2007). These results enabled the development team to assess the potential benefits of each technique, as well as the trade-offs needed for production code. We also

examined software engineering tools for GIS. One of the conclusions from this was the usefulness of *design patterns* and *generic programming* as a basis for achieving reuse in GIS software development (Câmara, Souza, Pedrosa, Vinhas, Monteiro, Paiva, Carvalho, and Raoult 2001; Vinhas, Souza, and Câmara 2002).

The last and most difficult problem is sharing the resulting code with the FOSS4G community. Many of the new tools and techniques might be unfamiliar for FOSS4G developers and practitioners. Hence there is a need to explain not only the code, but also the ideas behind it. Experience has shown that face-to-face workshops and meetings are the best way to discuss new ideas and their implementation. A second-best alternative is writing detailed documentation, which is not easy to achieve in open source projects (see Chapter 2 in this text). Developers have to work hard to share their results, and the TerraLib team is aware of this challenge.

3. THE DESIGN OF TERRALIB

This section discusses the requirements and design rationale for TerraLib. It presents the alternatives considered at various points, and explains the final choices. The discussion explains how product requirements led to the software architecture, the conceptual model and to the extensions to the basic OGC specifications.

3.1 Product Requirements

The main goal for TerraLib led to the following needs:

(a) *Ease of customization*: developers should require little effort to use the library to develop their applications. They should concentrate only on specific user needs, and the library should provide powerful abstractions that cover the common needs of a GIS application.

(b) *Upward compatibility to the OGC simple feature data model*: considering the impact and popularity of the OGC specifications, a TerraLib spatial database should be compatible with the OGC simple feature specification (SFS). This was not an original project requirement. When the project started in 2002, the authors underestimated the impact and extent of the OGC. TerraLib's code was redesigned (from version 3.2 to version 4.0) to satisfy this need.

(c) *Decoupling applications from the DBMS*: the library should handle different object-relational databases transparently.

(d) *Support for large-scale applications*: to be useful for environmental and socio-economic application, the library should provide efficient storage and retrieval of hundreds of thousands of spatial objects.

(e) *Extensibility*: a GIS library should be extensible by other programmers and introducing new algorithms and tools should not affect already-existing code.

(f) *Enabling spatio-temporal applications*: emerging GIS applications need support for different types of spatio-temporal data, including events, mobile objects, and evolving regions.

(g) *Remote sensing image processing and storag*e: the library should be able to handle large image databases, and inclusion of image processing algorithms should be easy.

(h) *Spatial analysis*: there should be support for spatial statistical methods, to improve our ability to extract information for socio-economic data.

(i) *Environmental modelling*: there should be support for environmental and urban models, including dynamic models using cellular automata.

To respond to issues (a) and (b), TerraLib has a strong conceptual model, as explained in Section 3.2. Points (c), (d) and (e) led to a software architecture described in Section 3.3. The last four issues are considered in Sections 3.4 to 3.7.

3.2 Conceptual Model

This section describes TerraLib's conceptual model, designed to support requirements (a) and (b) above. When designing TerraLib, the developers had to make choices which are typical of software library design (Krueger 1992; Fowler, Kom, and Vo 1995; Meyer 1990). Apart from general principles such as applicability, efficiency, ease of use, and ease of maintenance, there are important trade-offs. Consider two opposing visions:

(a) *Vision 1***:** Libraries should take a minimalist approach. They should provide only primitive building blocks and include generators that can combine these blocks to yield complex custom applications. They should be split into independent modules, with as few dependencies as possible. The developer's focus can be narrowed to those modules that are of interest (Batory, Singhal, Sirkin, and Thomas 1993).

(b) *Vision 2*: Libraries should have strong ideas behind them. All the functionalities and modules should work well together. The idea is to maximize reuse by minimizing cognitive distance. Krueger (1992, p. 136) defines this term as follows: "Cognitive distance is the amount of intellectual effort expended by software developers to take a software system from one stage to another". In this vision, the intellectual effort that software developers need to take a library and develop applications should be minimal. Application programmers use higher-level abstractions to build applications, and do not need to understand details of the library's source code.

The choice between the two visions depends on the library's design goals. For libraries designed to be part of larger software, the first vision is the usual choice. Examples are libraries such as the Standard Template Library (STL) in C++ (Austern 1998) and the *shapelib* utility for GIS (Warmerdam 2007 - see Chapter 5 of this book). At the other extreme, we find libraries designed to be easily extendible to build complete applications. One example is libraries that use the model-view-controller (MVC) pattern (Krasner and Pope 1988) such as Java Swing (Elliott Eckstein, Loy, Wood, and Cole 2002). Libraries with strong concepts dictate how the user should develop the application.

TerraLib follows the second vision, since it aims to make it easy for programmers to develop end-user applications. To do this, the library needs to consider the semantic mismatch between relational databases and object-oriented applications. Relational databases store information in *tuples*, but GIS applications manipulate *objects*. A typical GIS application consists of four steps: (a) querying the spatial database; (b) converting the query results (tuples) into objects; (c) manipulating these objects to create new objects; (d) displaying the resulting objects. Thus, applications need to distinguish between data sources (the *spatial database*) and data targets (*the set of objects that must be manipulated and displayed*). To reduce the cognitive distance from an OSS code to a deliverable application, the GIS developer needs a library that provides abstractions both for the data sources and for the data targets. These abstractions should support the four basic GIS components (query, conversion, manipulation and display). Consequently, TerraLib's supports the following abstracts:

- *Database*: a repository of information that contains data and metadata.

- *Layer*: a container of spatial objects that share a common set of attributes. Examples of layers are thematic maps (soil or vegetation maps), cadastral maps (map of land parcels in a city) or raster data such as satellite imagery. A layer knows its cartographical projection. Layers are inserted in the database by importing data from files or other databases, or by processing other layers. A layer stores the temporal evolution of objects it contains.

- *Representation*: the geometric parts of data contained in a layer. TerraLib supports different geometries, including two-dimensional (2D) vectors (points, lines or areas), cell spaces, networks, triangulated irregular networks (TINs), and multi-dimensional rasters. The same data can have different representations (for example a city can be represented by the polygon that describes its political boundaries or by a point that represents its geometric centre).

- *Theme*: a theme contains a subset of the objects of a layer, produced by a selection. The selection may use attribute, spatial or temporal conditions. Each theme has a set of presentation attributes for graphical display.

- *View*: this is a set of themes that are visualized or processed together. It defines a particular user's view of the database. A view has a unique cartographical projection, and the themes it contains are converted to this projection.

- *Visual*: this comprises a set of presentation attributes. Each theme has a unique visual. A visual includes filling and contour colours for polygons, thickness and colours for lines or symbols for points.

TerraLib distinguishes between data sources and data targets. The abstractions of *database*, *layer* and *representation* relate to the source domain and describe data organization and hierarchy. The ideas of *theme*, *view* and *visual* relate to the target domain and describe data retrieval and presentation. A query in TerraLib retrieves tuples from layers, converts these tuples into a set of objects, and groups objects of the same type in themes. Thus, *layer* and *theme* are complementary abstractions. *Layers* organize spatial data in the database. *Themes* organize objects for manipulation and display. Similarly, *databases* and *views* are complementary concepts. A *database* organizes *layers* of spatial data. A *view* organizes *themes* containing spatial objects.

These concepts provide a set of higher-level abstractions on top of the OGC Simple Feature Specification (SFS), which are not part of the current OGC model. Terralib stores these entities in a set of metadata tables, built when creating a new database. These metadata tables are kept updated as long as TerraLib manages the database. Should an OGC-compliant application access a TerraLib database, it will only access the tables described in the OGC model.

3.3 Software Architecture

This section discusses how TerraLib responds to requirements (c), (d) and (e) as stated in Section 3.1 (*DBMS-independence, efficiency* and *extensibility*). To address these issues, one important early decision was the choice of the C++ programming language. The developers had previous experience and many algorithms available in C++, as part of SPRING, their earlier GIS project (Câmara, Souza, Freitas and Garrido 1996). Existing DBMS such as PostgreSQL provide native interfaces in C++. Also, C++ helps with the use of generic programming (Alexandrescu 2001) and design patterns (Gamma, Helm, Johnson, and Vlissides 1995), which are powerful programming styles.

The developers chose an architectural design that has a kernel and a periphery. Maintenance of the kernel is the responsibility of a core team composed of a few senior programmers. Other contributors use the library's core to add new algorithms that test the library's core for extensibility and robustness. This follows the approach used for successful open source products such as Linux, PostgreSQL and Apache, which have a kernel whose maintenance is the responsibility of a small team. Contributions from the community occur at the external layers. As an example, out of more than 400 developers, the top 15 programmers of the Apache Web server contribute 88% of added lines (Mockus, Fielding, and Herbsleb 2002). TerraLib's architecture has four parts, as shown in Figure 12.1:

(a) *Kernel:* the core of TerraLib provides a set of spatio-temporal data types, code for cartographic projections and topological spatial operators, an API for storage and retrieval of spatio-temporal objects in databases, and classes for controlling visualization of spatial data.

(b) *Drivers*: modules that specialise the kernel's generic database API to allow access to DBMS such as PostgreSQL (with and without the PostGIS spatial extension) or MySQL, and to external files in both open and proprietary

formats. Basic maintenance and upgrade is the responsibility of the project core team.

(c) *Functions:* algorithms that use the kernel application programming interface (API). Typical functions include spatial analysis and query and simulation languages. The functions are designed to allow external contributions.

(d) *Interfaces*: consist of different interfaces to the TerraLib library to allow software development in different environments (Java, COM) and the support for OGC services such as WMS, WFS and WCS.

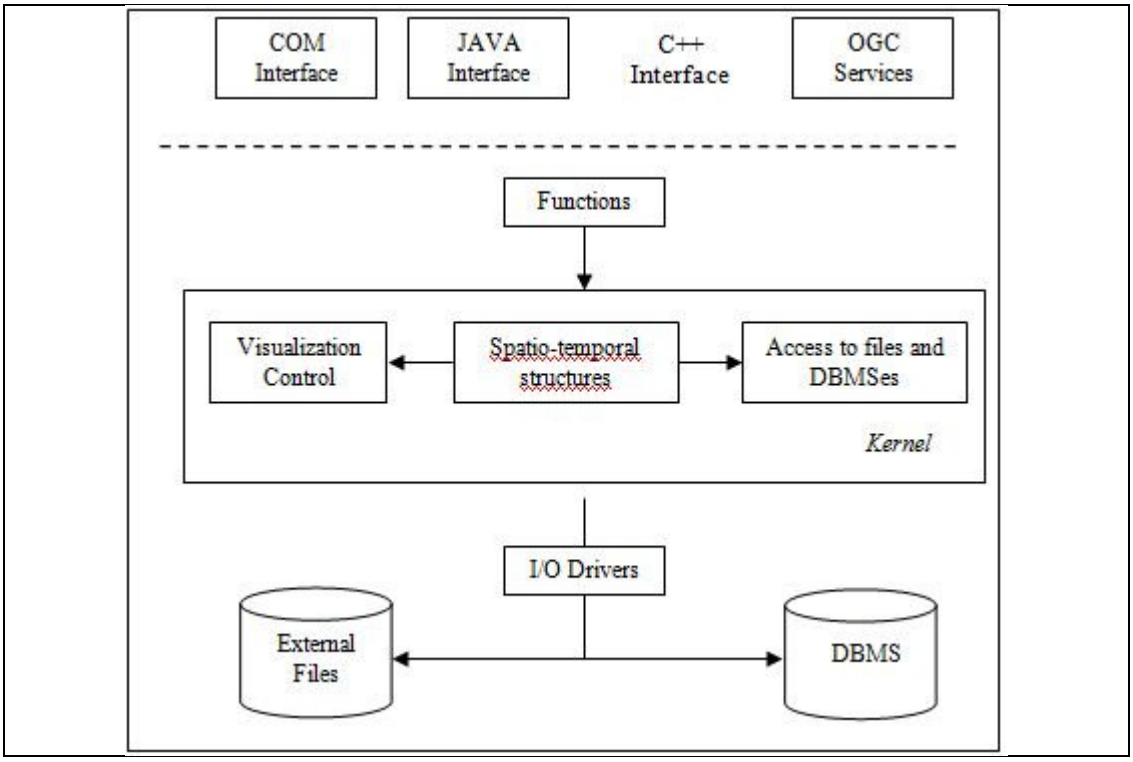

Figure 12.1. TerraLib software architecture.

The core of TerraLib's kernel is its set of spatio-temporal data types and its methods for query processing. The OGC Simple Feature Geometry (OGC SFS) model is used for storing basic vector geometries. Extra metadata tables support the abstractions described in Section 3.2. For a full description of the metadata, see the library's documentation (Vinhas, Ferreira, and Ribeiro 2007). One key need is efficiency. The developers have spent much effort on issues such as indexing techniques and computational geometry algorithms (Queiroz, 2003; Rodrigues, Cavalier, Andrade, and Queiroz 2005; Rodrigues, Andrade,

Queiroz, and Magalhães 2006). With this work, the library supports large-scale geographical databases, as discussed below in Section 4.

The I/O drivers provide the interface between the kernel and the various DBMS and file formats. The library handles different object-relational databases, using a generic database API that handles specific features of each DBMS. Using this API, the TerraLib programmer works at an abstract level. TerraLib hides the differences between products such as PostgreSQL/PostGIS and MySQL from the programmer (Ferreira et al., 2002).

The design of TerraLib *functions* aims for extensibility, as introducing new algorithms and tools should not affect existing code. Adoption of the principles of generic programming and design patterns helped achieve extensibility (Câmara et al., 2001). Three design patterns were found to be especially useful:

(a) *Factory*: this pattern provides an interface for creating an object, but lets subclasses decide which class to instantiate. In GIS, it is useful to include new functions without changing existing code. For example, there are hundreds of cartographic projections. When code for a new projection is inserted in TerraLib, it tells the projection factory about its existence. The projection factory will call this new code when needed.

(b) *Strategy*: provides an interface to a family of algorithms, and makes them interchangeable. In GIS, the strategy pattern is useful when there are different ways of performing the same function. This occurs often in image processing. For example, there are many different types of image filters. Using the strategy pattern, the programmer can use the same code for different filters.

(c) *Iterators:* TerraLib uses iterators to decouple algorithms from data structures. For example, for computing a histogram it is not essential to know if the data are a set of points, a set of polygons, a grid or an image. All the algorithm needs is to be able to look into a list and to get the values of the items that satisfy a certain property (for example, those that are closer in space than a specified distance). In a similar way, most spatial analysis algorithms can be independent of spatial data structures and described only by their properties (Vinhas et al., 2002).

3.4 Raster Data Handling

As noted earlier, TerraLib handles raster data types as well as vector data. All raster data types are handled in a unified way, using an interface with two main methods: one to set the value of a point on a multidimensional raster and one to recover its value. The library provides decoders for different raster data formats, and iterators for accessing image data and developing image processing algorithms. The decoders are responsible for handling the particularities of each data source. The iterators are specialized pointers that traverse a raster in a predefined way (for example, only inside a given polygon). They hide the internal details of the raster data (Vinhas et al., 2003).

The library has drivers for storing raster data in different DBMS. TerraLib uses indexing and compression to achieve good performance in a standard DBMS, even for large satellite images. Indexing combines tiling and multi-resolution pyramids. Multi-resolution pyramids store the raster data at various sizes and degrees of resolution. Each resolution level is divided in tiles. A tile has a unique bounding box and a unique spatial resolution level, which are used to index it. Figure 12.2 shows a pictorial representation of the tiling and multi-resolution storage model.

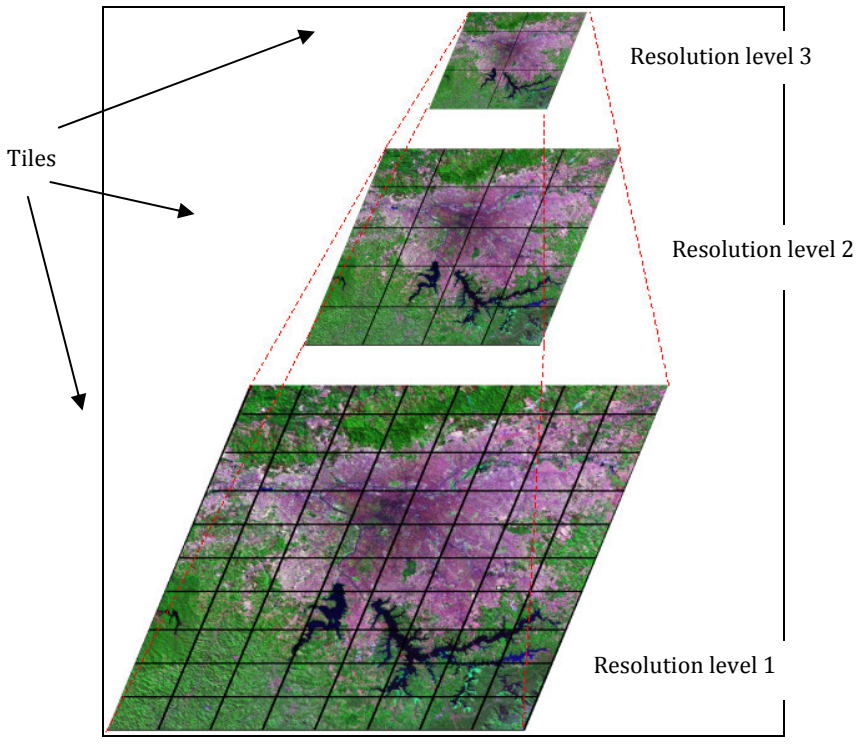

Figure 12.2. TerraLib raster data indexing.

The multi-resolution pyramid approach is useful for display of large data sets, avoiding unnecessary data access. TerraLib stores the whole pyramid. To compensate the extra storage needs, it applies lossless compression to the individual tiles. When retrieving a section of the data, only the relevant tiles are accessed and decompressed.

TerraLib provides a large set of image processing algorithms including filters, segmentation, classification, mixture models, and geometric transformations. The image processing algorithms have a common interface receiving as input instances of the raster API and a set of parameters. Two particular important algorithms are the object-oriented segmentation and region classifier developed by INPE, originally part of the SPRING software (Câmara et al 1996). This algorithm has been extensively validated for extracting land use patterns in tropical forests and was favourably reviewed in a recent survey (Meinel and Neubert 2006).

Creating new image processing algorithms is straightforward. The library has a set of standard protocols that combine the '*Factory*' and '*Strategy*' design patterns. First, the programmer develops his new algorithm (for example, a filter). Then, he signals that TerraLib should use his proposed strategy to filter the image. The programming manual provides further details (Vinhas et al. 2007).

3.5 Spatio-temporal Queries

There are different ways to record spatio-temporal information in a database. The main alternatives are: (a) providing snapshots of data; (b) storing sequences that describe the temporal evolution of spatial objects; (c) storing both objects and events that change them (Hornsby et al., 2000; Grenon and Smith, 2003; Galton, 2004; Worboys, 2005). TerraLib adopts a dual perspective, archiving fields (stored in raster data) as snapshots and objects (stored in vector data) as sequences. A spatio-temporal object in TerraLib is a sequence of static objects with the same identifier. Each static object is valid for one interval.

Storage and retrieval of spatio-temporal objects in DBMS needs more abstractions beyond those discussed in section 3.2. TerraLib distinguishes four types of data stored in layers: static (unchanging data), *events* (singular occurents such as crimes), *moving objects* (such as cars in highways) and *evolving objects* (such as cities whose boundaries and attribute values change in

time). All spatio-temporal objects that share the same attributes are converted to tuples of a layer (including timestamps) and stored together in a database. Metadata tables store information about different types of layers and point out which attributes of a layer store the timestamps associated to the temporal intervals. For example, consider an urban cadastre. All lots of a city for all intervals are stored together in a single layer. Grouping all objects together simplifies data handling. Inside the database, TerraLib uses optimization techniques for dealing with large data volumes.

As discussed in section 3.2, a GIS application needs to transform database tuples (data sources) into objects that can be manipulated and displayed (data targets). In a purely static and non-temporal GIS, it is enough to use *themes* to group objects of the same type resulting from queries. In this case, all objects contained in a theme belong to the same interval.

When a *theme* of spatio-temporal data is retrieved from the database, it contains all spatio-temporal objects for the whole period when data is available. However, not all objects exist in all instances. Consider the case of real estate lots in an urban cadastre. Extracting data from the 'lots' *layer* will produce a *theme* composed of all lots that ever existed in the cadastre. An application may need to use only those lots that currently exist. In this case, we need to perform a spatio-temporal selection inside a *theme*. To do this, TerraLib uses an extra concept: a spatio-temporal object (*STObject*). An STObject is an individual entity that preserves its identity, but may change its location and the values of its attributes.

TerraLib provides a query processor to extract *STObjects* from themes (Ferreira et al. 2005). The query processor has a generic API for programmers, hiding data storage details. Algorithms can handle spatio-temporal data using only *STObjects* returned by the query processor.

To perform a query, the application programmer defines three different restrictions (spatial, temporal, and attribute). The spatial restrictions use the OGC-specified topological predicates and the temporal restrictions use Allen's interval predicates: *before, meets, overlaps, finished, during, starts,* and *equals* (Allen, 1983). Combining of these restrictions, the query processor is able to response questions as: *"For each month, which changes occurred in the parcel?"*, "*Which crimes happened on Friday in the south zone of Rio de Janeiro?*" and *"What was the path followed by this wolf in July of 2006?"*.

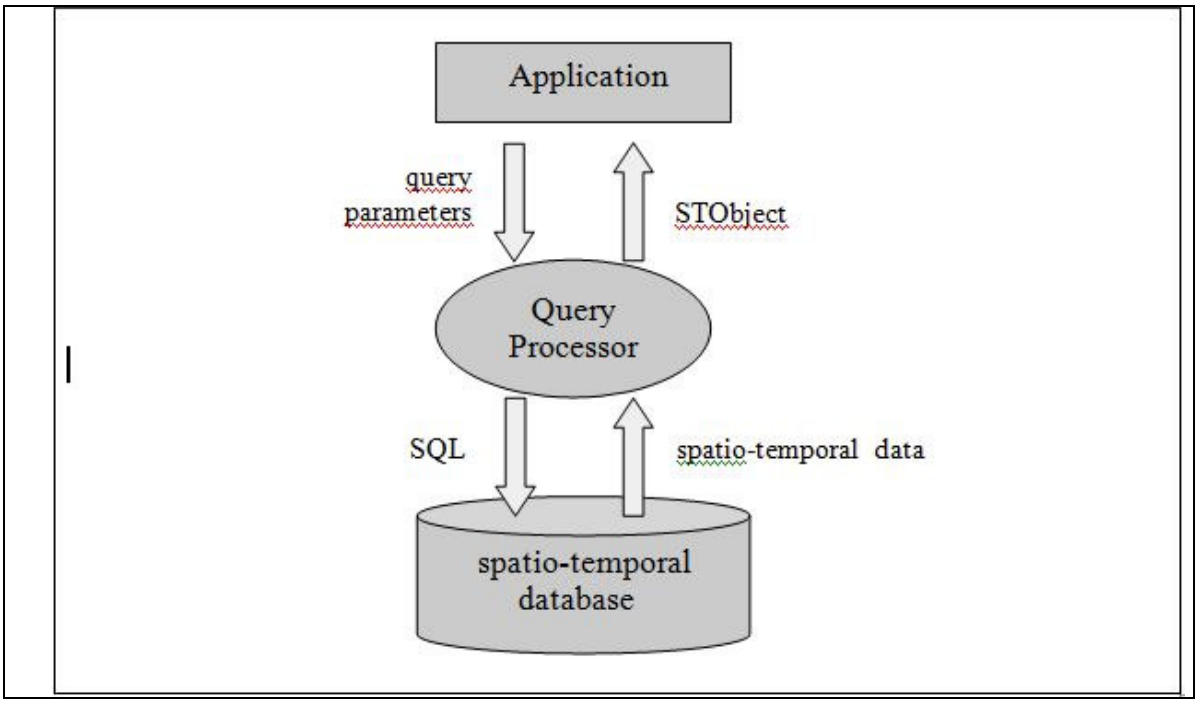

Figure 12.3. The spatio-temporal data query processor (source: Ferreira et al. 2005)

3.6 Spatial Statistics in TerraLib

A GIS produces color maps of variables such as individual populations, quality of life indexes or company sales in a region. However, to explore the underlying information present in the data, visualisation is not enough. To make effective use of environmental and socioeconomic data, a GIS should provide statistical methods and models. *Spatial statistics* methods measure properties and relations and translate the existing patterns into objective measures. They include geostatistics (Goovaerts 1997), global and local autocorrelation indexes (Anselin 1995), analysis of point patterns (Diggle 2003), regionalization (Openshaw et al. 2001; Martin 2003) and spatial regression (Anselin 1988).

One approach to link spatial statistics tools to GIS is to use loose coupling mechanisms, where the GIS does data conversion and graphic display, and the spatial models run separately. Examples are the links between SpaceStat and ArcView (Anselin and Bao 1997) and between R and Grass (Bivand and Neteler 2000). A more recent trend is to integrate spatial statistics methods in a GIS. An example is GeoDa (Anselin, Syabri, and Kho 2006) that provides a graphical user interface for exploratory spatial data analysis on points and polygons.

TerraLib has a basic spatial statistical package, including local and global autocorrelation indexes, non-parametric kernel estimators, and regionalization methods (Assunção et al. 2006). These functions can be used by GIS applications. One such application is TerraView, described in the next section. Additionally, TerraLib provides a direct link with the R programming language using the *aRT* package (Andrade and Ribeiro 2005). R is an open source programming language for statistical computing and graphics and has become a *de facto* standard for developing statistical software (Ihaka and Gentleman 1996). R has contributors from all over the world, with continuous improvement that incorporates cutting-edge statistical methods. Integration with R can keep a GIS always updated with recent research on spatial statistics. Packages in R relevant to GIS include *geoR* for geostatistics (Diggle and Ribeiro 2007), *splancs* for analysis of point processes (Rowlingson and Diggle 2003), and *sp* that provides general support for spatial analysis (Pebesma and Bivand 2005).

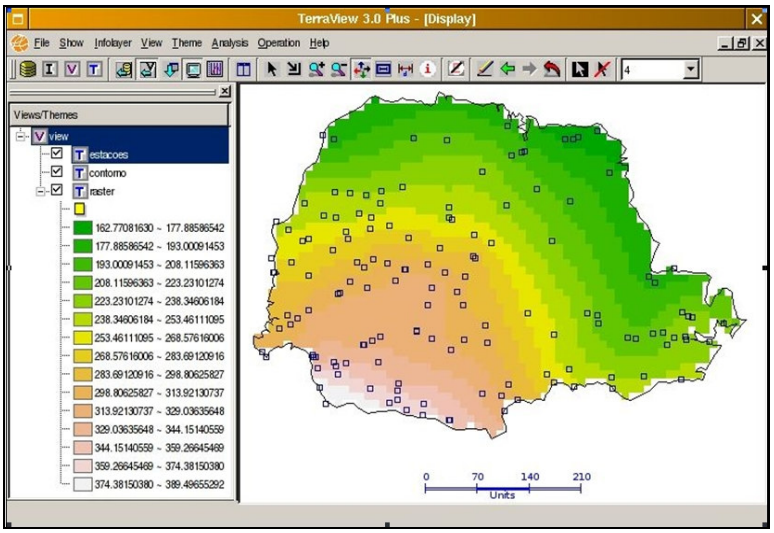

Figure 12.4 Plotting the result of an R algorithm in TerraView (Andrade et al. 2005).

The *aRT* API performs spatial queries and operations in R. It encapsulates TerraLib functions into R objects, and enables R users to read data from a TerraLib database. This coupling satisfies three requirements:

(a) Statisticians can implement methods of data analysis using R and call TerraLib's facilities for data storage and computational geometry directly for R;

(b) TerraLib programmers can quickly develop interfaces for calling wrapped R code, which consists of functions and a description of their arguments. They do not need to know about R internals.
(c) Users of TerraLib-based applications can perform data analysis in R, without knowing the R syntax or even without noticing their analysis are executed by R.

An example of the R-TerraLib coupling is shown in Figure 12.4. The base data is a set of point samples stored in a TerraLib database. This data was interpolated into a grid using the *geoR* package (Diggle and Ribeiro 2007) and the result stored as a TerraLib layer and displayed using the TerraView GIS application.

3.7 Cell Spaces and Cellular Automata

A cell space is a spatial data type where each cell handles one or more types of attribute. Cell spaces were part of early GIS implementations (Dutton 1978) and later discarded by one-attribute raster data structures, mostly because of efficiency issues. We believe it is time to reconsider this decision and to reintroduce cell spaces as a basic data type in a GIS. Cell-spaces have several advantages over raster-based layers as a means of storing information about continuous spatial entities. Using one-attribute raster data to store results for dynamical models requires storing information in different files; this separation results in increased complexity in data management and user interface. A cell-space stores all attributes of a cell together, with significant benefits for modeling.

Cell-spaces have been used in the last two decades for simulation of urban and environmental models, as part of cellular automata (CA) models (Batty 2000). Most CA models link to a GIS by loose coupling mechanisms, where the GIS performs the data handling and graphic display, and the spatial models run outside the GIS database. There is extra work for data translation, and possible problems of redundancy and consistency. Modeling tools also lack GIS spatial analytical capacities. To address these drawbacks, cell-space models need strong links to the GIS architecture. Using strong coupling, modeling and GIS can be made more robust through their linkage and co-evolution (Parks 1993).

TerraLib supports cell spaces as one of its native data types. It provides functions for storage and retrieval of cell spaces in a DBMS and algorithms for creating cell spaces from vector data. Cell spaces enabled development of the TerraME language, which is an add-on to TerraLib that enables simulation in 2D

cellular spaces. It supports multi-scale spatial models, where each scale has a different extent and resolution (Carneiro 2006).

Two important innovations in TerraME are its use of anisotropic spaces (Aguiar, Câmara, and Cartaxo 2003) and of hybrid automata models (Henzinger 1996). Anisotropic spaces arise when modeling natural and human-related phenomena. For example, land settlers in a new area do not occupy all places at the same time. They follow roads and rivers, leading to an anisotropic pattern. However, most spatial statistical and dynamical modeling techniques fail to incorporate spatial anisotropy, leaving out spatial relations that are variable over space. This leads to a serious challenge in producing models that approximate reality, since most real-file spaces are anisotropic.

A *hybrid automaton* is an abstract model for a system whose behavior has discrete and continuous parts (Henzinger 1996). It extends the idea of finite automata to allow continuous change to take place between transitions. Adopting hybrid automata in spatial dynamical models allows complex models which include critical transitions. Inside each discrete state, the model variables can change. When a critical value happens, the model moves from the current state to a new one, which is governed by different equations. For example, consider a model for tropical vegetation that has a critical threshold caused by land use change. Under conditions of small land change, the vegetation follows one growth model. When a critical condition is reached, we need to use a different growth model. The use of a hybrid automaton allows modeling the tropical vegetation under two different conditions (Carneiro 2006).

Among the typical applications of TerraME are land change and hydrological models. Figure 12.5 shows an application of TerraME for land use change modeling in the Brazilian Amazon. The scenario considers the possible impact of paving a road between the cities of Porto Velho and Manaus, This road crosses areas of currently pristine tropical forest. The model results provide an estimate of how much increase in deforestation could occur in the region from 1997 to 2020 (Aguiar, 2006). These models have proven to be useful for supporting public policies that protect the environment and aim at sustainable development practices.

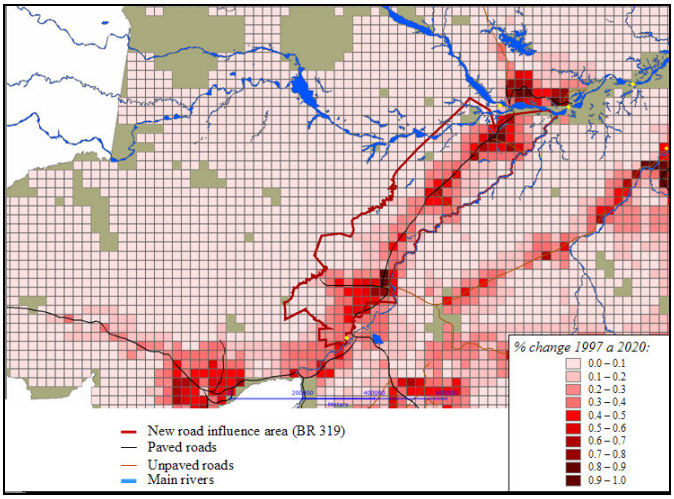

Figure 12.5 Spatial modeling of projected deforestation of a new road in Amazonia from 1997 to 2020 (source: Aguiar, 2006).

4. DEVELOPMENT OF GIS APPLICATIONS USING TERRALIB

This section provides an outline on how to develop a GIS application using TerraLib and describes selected GIS applications that use TerraLib. We describe the general principles of GIS application development and then describe of TerraView, an open source GIS for spatial data analysis, and TerraAmazon, Brazil's national database for monitoring deforestation.

4.1 Building an Application Using TerraLib

TerraLib provides C++ classes that support the higher-level abstractions described in section 3, such as *Database, Layer, View* and *Theme*. When a programmer builds an application using TerraLib, he should use them. In this section, we provide a brief guide to the steps involved in GIS application development. On what follows, TerraLib classes are in `monospaced` font (e.g., **`Database`** is the TerraLib class for a spatial database). For a more details, see TerraLib's programming tutorial (Vinhas et al. 2007).

Consider first the case of a GIS application using static data, which has four steps: (a) querying the spatial database; (b) converting the query results (tuples) into objects; (c) manipulating these objects to create new objects; (d) displaying the resulting objects. To perform these steps in TerraLib, the programmer should include code that does the following:

1. Choose the DBMS that will support the application.
2. Create a TerraLib **`Database`**.
3. Connect to the **`Database`**.

4. Import data to create a new **`Layer`** from standard formats.
5. Create a view to store a user's view of the database using **`View`**.
6. Create a **`Theme`** and insert it to the user's **`View`**.
7. Define the contents of the **`Theme`**, by pointing to the data source (a **`Layer`**) and defining attribute and spatial restrictions over that **`Layer`**.
8. Load the contents of the **`Theme.`**
9. Define the display parameters of the **`Theme`** using a **`Visual.`**
10. Display the **`Theme`** using a GUI toolkit.

A second situation arises when the GIS application uses spatio-temporal data. Then, after step 8, the developer should include code for the following operations:

9a. Create a **`Querier`** (query processor).

10a. Define the spatial, attribute and temporal restrictions to apply the query using **`Querier`**.

11a. Apply the query and get an STObjectSet (set of spatio-temporal objects).

12a. Manipulate and display the **`STObjectSet`**.

These steps show that Terralib abstractions encapsulate a general view of how a GIS works. The abstractions of *database*, *layer*, *theme* and *view* are especially important, as they provide a link between what is stored in a database and what is selected and manipulated by the user. Thus, a *database* organizes spatial data in *layers*; a user manipulates *themes*, according to his *view* of the database. Layers store different types of spatio-temporal data and thus provide the containers needed by the database.

Although these concepts are not part of the current OGC specifications, they arise from decades of experience. Most commercial and open source GIS application use them implicitly or directly. Building a graphical user interface on top of these abstractions is simple. A graphical user interface creates events that match to actions that call TerraLib functions. If the user knows the TerraLib ideas, each of these actions will consist of small pieces of code, based on standard examples.

4.2 Examples of Open Source Applications

TerraView is an open source GIS for spatial data analysis, which provides the basic functions of data conversion, display, exploratory spatial data analysis, spatial statistical modeling, and spatial and non-spatial queries. This product is a general-purpose GIS for TerraLib databases. Many Brazilian public institutions use TerraView for public policy making, including studies in spatial epidemiology and crime analysis. Figure 12.6 shows TerraView's user interface.

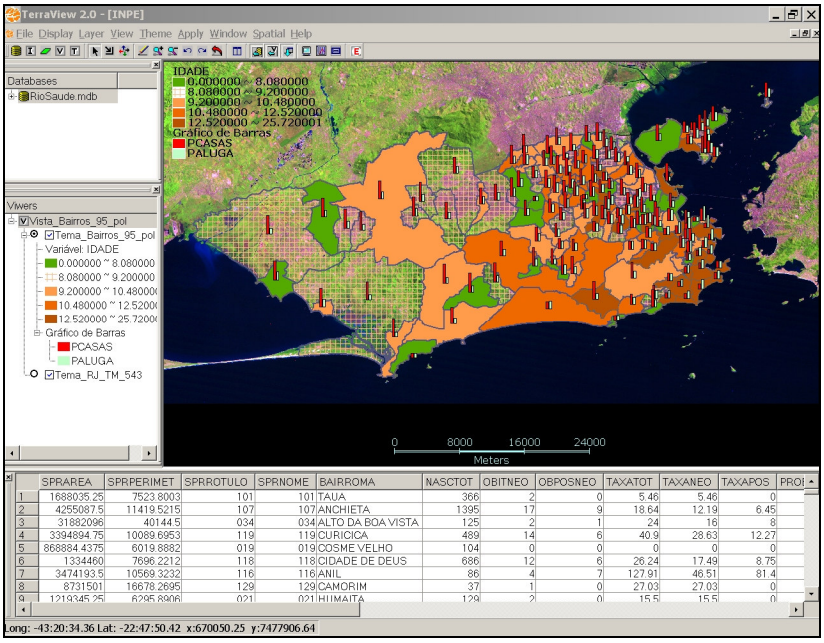

Figure 12.6 User Interface for the TerraView product.

TerraView is open source software licensed using the GPL. Its graphical user interface uses the QT cross-platform framework (Blanchette and Summerfield 2006). Programmers can extend its functionalities in two ways. By adapting the menus, they can include new functionalities or change behaviour of existing ones. Additionally, TerraView supports plug-ins, which are independent applications that can access a TerraLib database. Using TerraLib, plug-ins have access both to the database and to the display controls: the lists of views, themes and its canvas. TerraView is available at `http://www.dpi.inpe.br/terraview/`.

A second noteworthy application is *TerraAmazon*, Brazil's national database for monitoring deforestation in Amazonia, developed by INPE and its partner, the Foundation for Space Science, Technology and Applications (FUNCATE). The DBMS is PostgreSQL version 8.2, running on a Linux server. The application

manages all data workflow, gathering about 600 satellite images, pre-processing, segmenting, and classifying these images for further human interpretation, in a concurrent multi-user environment (see Figure 12.7). The database stores about 2 million complex polygons and grows yearly with 60 gigabytes of full resolution satellite images, using the TerraLib pyramidal resolution schema. A Web site allows seamless display and analysis of full resolution data, using TerraLib's PHP extension and TerraLib's OGC WMS server.

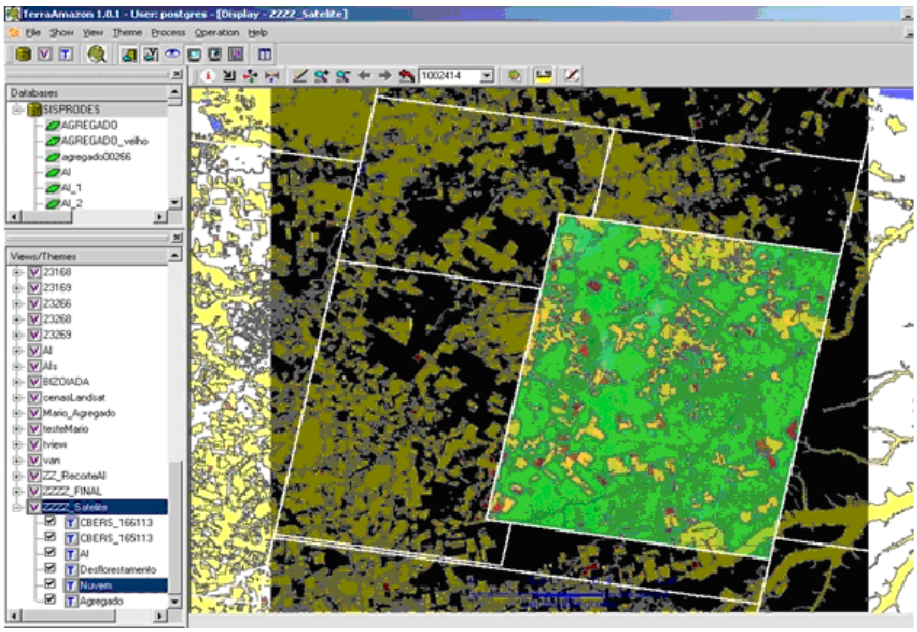

Figure 12.7 User Interface for TerraAmazon.

5. Licensing and Maintenance Policy

One of the important decisions on the TerraLib project was to decide on its license and long-term maintenance policy. The decision considered the nature of the GIS market and the strategy for open source technologies to reach a critical mass of users. The GIS software market is an oligopoly where ESRI, Bentley and Intergraph have a market share greater than 50% (Daratech, 2006). This leads to a "vendor lock-in" effect (Arthur, 1994). The "lock-in" effect occurs when a customer is dependent on a vendor for products and services and cannot move to another vendor without large switching costs. There are many causes for vendor dependence and reluctance to use FOOS4G. First, commercial GIS products use proprietary data formats, making users apprehensive to the

costs of data conversion. Secondly, each GIS adopts a different data model and user interfaces, which needs much training for effective use. Finally, the user worries about long-term maintenance of his archive and his applications, as he fears for the sustainability of FOSS4G projects. This results in a conservative policy for most GIS adopters.

Service providers based on open source software face a tough challenge. Convincing a user to change from a commercial to an open source product is a big effort. Users will consider carefully the risks involved in choosing an open source solution, compared to well-publicized commercial products. Software cost is only part of the problem. Users worry about long-term assurance to protect their investments on data capture and on specialized applications. To convince prospective customers, service providers using FOSS4G need to build custom applications fast and reliably, a task that needs investment. Most service providers consider these applications as intellectual property that needs protection. Thus, they are unwilling to invest in open source software that has binding licenses, such as the GPL (GNU Public License).

When deciding on the TerraLib license, the authors considered there should be a strong incentive for commercial companies to use the library to reduce the "vendor lock-in" effects of the GIS market in Brazil. Thus, we chose to release TerraLib according to the Lesser GNU Public License (LGPL). The LGPL allows private companies to build their applications on top of TerraLib, and market them as proprietary software, while the TerraLib software itself remains publicly licensed. A second consideration involves development and maintenance of the TerraLib kernel. The Brazilian government has guaranteed its long-term support for the core team of developers. INPE provides capacity building for developers, and supports service companies that use the software. By mid-2007, there are about 10 Brazilian services providers using TerraLib for building commercial applications. Each company has a market focus, including utilities, oil industry, agriculture, urban cadastre, and the military. There is evidence that this strategy is paying off. Companies offering GIS services based on TerraLib form 10% of the service provider market in Brazil. The impact on the commercial market is an indicator of a decrease in the "vendor lock-in" effect. We consider the library's licensing and maintenance policy are an essential part of this result.

An example of a proprietary application that uses TerraLib is InfoPAE, developed by the Computer Graphics Group (TecGraf) of the Catholic University in Rio de Janeiro (PUC-RIO) in partnership with Petrobras (Petróleo Brasileiro S.A.) a Brazilian oil company. The application was designed for emergency response on the oil industry. InfoPAE works with local emergency action plans (LEAPs) that handle significant events. An LEAP is an organized collection of actions, similar to a workflow, coupled with information stored in geographical as well as conventional databases. LEAP frameworks are useful to design large emergency plans. InfoPAE has is being used in more than 100 installations of Petrobras in Brazil.

6. CONCLUSIONS

The design and implementation of TerraLib serve an example of the challenges involved in building a FOSS4G library that allows innovative applications and supports large-scale applications. One of main lessons learned is that the current set of OGC specifications is not enough to support these goals. Thus, TerraLib has introduced extra abstractions to reduce the cognitive distance between the GIS developer and the application. These extra abstractions come at a price. When a FOSS4G developer adheres strictly to the OGC specifications, his/her code runs in all OGC-compliant libraries. Adopting TerraLib reduces GIS application development effort, at the cost of using abstractions not supported by other products. OGC-compliant applications will be able to access the part of a TerraLib database that contains OGC's simple features. But the extra relations used by TerraLib to handle abstractions such as *theme* and *view* and data structures such as *cell spaces* will be invisible to these applications.

When introducing solutions for spatio-temporal queries, cell spaces and raster data handling, we had to make choices. Only further experience will show if we made the right decisions. Another difficult issue concerned software architecture and design for extensibility. The extensive use of design patterns in TerraLib suits experienced programmers who are comfortable with ideas such as 'factory' and 'strategy'. Novice developers need at least six months training in C++ before they can become skilful in these concepts.

To sum up, developing TerraLib has shown how difficult it is to design a GIS library that combines simplicity and expressiveness. The developers opted for expressiveness at the expense of simplicity in this case. This choice may limit

the rate of adoption of TerraLib by the FOSS4G community. Nevertheless, it is the developers' hope the library's assets may be attractive to other developers that want to build large-scale GIS applications.

Acknowledgments

TerraLib's core team, apart from the authors, includes Laercio Namikawa and Emiliano Castejon at INPE. *TerraView* has been designed and implemented by Juan Pinto de Garrido and Lauro Hara. Additional contributors to TerraLib include Tiago Carneiro, designer of *TerraME*, Pedro Andrade, who developed *aRT*, Ana Paula Aguiar, who wrote code for cell spaces, and Felipe Castro da Silva and Thales Korting, who developed image processing functions. The technical support of Julio D'Alge on cartographical projections and Leila Fonseca in image processing has been important. We also have important contributions from Paula Frederick, Marcelo Metello, Natacha Barroso and Leone Pereira Masieiro at PUC-Rio, and Rui Mauricio Gregório and Vanildes Ribeiro at FUNCATE. The TerraLib project is partially financed by CNPq grant no. 552040/02-9. Gilberto Câmara's research is also financed by a CNPq grant no. 300557/96-5. The project has also received financial support from FAPESP (Fundação de Amparo à Pesquisa no Estado de São Paulo).

TerraLib code relies on a number of open source software packages. INPE and TerraLib development team thanks the open source community for their efforts. These libraries are: (a) the *zlib* library to compress data when storing raster data in a TerraLib database; (b) the independent JPEG Group's library for JPEG image compression; (c) *libgeotiff* to decode/encode raster data in TIFF/GEOTIFF format; (d) *shapelib* to decode/encode vector data in shapefile format.

As of mid-2007, TerraLib consists of 280.000 lines of C++ code and 170.000 lines of third-party open source utilities. The development started in 2001, with an effort of 60 man-years spent so far in TerraLin and TerraView. The library and associated applications may be obtained from the website `http://www.terralib.org`.

References

Aguiar, A., Câmara, G. and Cartaxo, R. (2003). Modeling Spatial Relations by Generalized Proximity Matrices. V Brazilian Symposium in Geoinformatics - GeoInfo 2003, Campos do Jordão, SP, Brazil.

Aguiar, A. P. D. (2006). Modeling Land Use Change in the Brazilian Amazon: Exploring Intra-Regional Heterogeneity. PhD Thesis, Remote Sensing Program. Sao Jose dos Campos, INPE.

Aksoy, S., Koperski, K., Tusk, C. and Marchisio, G. (2004). Interactive Training of Advanced Classifiers for Mining Remote Sensing Image Archives. ACM International Conference on Knowledge Discovery and Data Mining, Seattle, WA, ACM.

Alexandrescu, A. (2001). *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison-Wesley, Reading.

Allen, J. F. (1983). Maintaining Knowledge about Temporal Intervals. Communications of the ACM 26, 832-843.

Almeida, C. M., Batty, M., Monteiro, A. M. V., Câmara, G., Soares-Filho, B. S., Cerqueira, G. C. and Pennachin, C. L. (2003). Stochastic cellular automata modeling of urban land use dynamics: empirical development and estimation. Computers, Environment and Urban Systems 27, 481-509.

Anselin, L. (1988). *Spatial econometrics: methods and models*. Kluwer, Dordrecht.

Anselin, L. (1989). What's Special about Spatial Data:Alternative Perspectives on Spatial Data Analysis. Santa Barbara, CA, NCGIA Report 89-4.

Anselin, L. (1995). Local indicators of spatial association - LISA. Geographical Analysis 27, 91-115.

Anselin, L. (1999). Interactive techniques and Exploratory Spatial Data Analysis. In: P. Longley, M. Goodchild, D. Maguire and D. Rhind (Eds.), *Geographical Information Systems: principles, techniques, management and applications.* Geoinformation International, Cambridge.

Anselin, L., Syabri, I. and Kho, Y. (2006). GeoDa: An Introduction to Spatial Data Analysis. Geographical Analysis 38, 5-22.

Andrade, P. R. and Ribeiro, P. J. (2005). A Process and Environment for Embedding The R Software into TerraLib. VII Brazilian Symposium on Geoinformatics (GeoInfo 2005), Campos do Jordao, Brazil, INPE/SBC.

Arthur, B. (1994). *Increasing Returns and Path Dependence in the Economy*. The University of Michigan Press, Ann Arbor, MI.

Assunção, R., Neves, M., Camara, G. and Freitas, C. d. C. (2006). Efficient regionalisation techniques for socio-economic geographical units using minimum spanning trees. International Journal of Geographical Information Science 20, 797-812.

Austern, M. (1998). *Generic Programming and the STL : Using and Extending the C++ Standard Template Library*. Addison-Wesley, Reading, MA.

Batory, D., Singhal, V., Sirkin, M. and Thomas, J. (1993). Scalable software libraries. SIGSOFT Softw. Eng. Notes 18, 191-199.

Batty, M. (2000). GeoComputation Using Cellular Automata. In: S. Openshaw and R. J. Abrahart (Eds.), *GeoComputation.* Taylor&Francis, London, 95-126.

Blanchette, J. and Summerfield, M. (2006). *C++ GUI Programming with Qt 4*. Prentice Hall, Indianapolis, Indiana.

Blaschke, T. and Hay, G. (2001). Object-oriented image analysis and scale-space: theory and methods for modeling and evaluating multiscale landscape structure. International Archives of Photogrammetry and Remote Sensing 34, 22–29.

Bivand, R. and Neteler, M. (2000). Open source geocomputation: using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems. 5th International Conference on GeoComputation, Greenwich, UK.

Burrough, P. (1998). Dynamic Modelling and Geocomputation. In: P. Longley, S. Brooks, R. McDonnell and B. Macmillan (Eds.), *Geocomputation: A Primer.* John Wiley, New York.

Câmara, G., Egenhofer, M., Fonseca, F. and Monteiro, A. M. (2001). What´s In An Image? Spatial Information Theory: Foundations of Geographic Information Science. International Conference, COSIT 2001, Santa Barbara, CA., Springer.

Câmara, G., Souza, R., Freitas, U. and Garrido, J. (1996). SPRING: Integrating Remote Sensing and GIS with Object-Oriented Data Modelling. Computers and Graphics 15, 13-22.

Câmara, G., Souza, R. C. M., Pedrosa, B. M., Vinhas, L., Monteiro, A. M. V., Paiva, J. A. C., Carvalho, M. T. and Raoult, B. (2001). Design patterns in GIS development: the TerraLib experience. III Simpósio Brasileiro de GeoInformatica, Rio de Janeiro, RJ.

Câmara, G., Neves, M., Monteiro, A. M. V., Souza, R. C. M., Paiva, J. A. and Vinhas, L. (2002). SPRING and TerraLib: Integrating Spatial Analysis and GIS. CSISS Specialist Meeting on Spatial Data Analysis Software Tools, Santa Barbara, CA, CSISS.

Carneiro, T. (2006). Nested-CA: a foundation for multiscale modeling of land use and land change. Computer Science Department. Sao Jose dos Campos, INPE. Doctorate Thesis in Computer Science.

Chang, S. K., Yan, C. W., Dimitroff, D. and Arndt, T. (1988). An Intelligent Image Database System. IEEE Transactions on Software Engineering 14, 681-688.

Couclelis, H. (1997). From Cellular Automata to Urban Models: New Principles for Model Development and Implementation. Environment and Planning B: 24, 165-174.

Daratech (2006). GIS Markets and Opportunities 2006 Survey. Cambridge, MA, Daratech Inc.

DeWitt, D., Kabra, N., Luo, J., Patel, J. and Yu, J.-B. (1994). Client-Server Paradise. VLDB Conference, Santiago, Chile.

Diggle, P. (2003). *Statistical Analysis of Spatial Point Patterns*. Edward Arnold, London, 2nd edition.

Diggle, P. and Ribeiro, P. J. (2007). *Model-based Geostatistics*. Springer, Heidelberg.

Dutton, G., Ed. (1978). *First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems*. Addison-Wesley, Reading, MA.

Elliott, J., Eckstein, R., Loy, M., Wood, D. and Cole, B. (2002). *Java Swing*. O'Reilly Press, Sebastopol, CA.

Erwig, M. and Schneider, M. (2002). Spatio-Temporal Predicates. IEEE Transactions on Knowledge and Data Engineering 14, 881-901.

Feitosa, F., Camara, G., Monteiro, A. M., Koschitzki, T. and Silva, M. S. (2007). Global and Local Spatial Indices of Urban Segregation. International Journal of Geographical Information Science 21, 299-323.

Ferreira, K. R., Queiroz, G., Paiva, J. A., Souza, R. C. and Câmara, G. (2002). A Software Architecture for Building Spatial Databases with Object-Relational DBMS. XVII Brazilian Symposium on Databases, Gramado, RS.

Ferreira, K. R., Vinhas, L., Queiroz, G. R., Câmara, G. and Souza, R. C. M. (2005). The Architecture of a Flexible Querier for Spatio-Temporal Databases. VII Brazilian Symposium in Geoinformatics, Campos do Jordao, Brazil

Fonseca, F., Egenhofer, M., Agouris, P. and Camara, G. (2002). Using Ontologies for Integrated Geographic Information Systems. Transactions in GIS 6, 231-257.

Fotheringham, A. S., Brunsdon, C. and Charlton, M. (2002). *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. Wiley, Chichester.

Fowler, G. S., Korn, D. G. and Vo, K.-P. (1995). Principles for writing reusable libraries. Proceedings of the 1995 Symposium on Software reusability. Seattle, Washington, United States, ACM Press.

Galton, A. (2004). Fields and Objects in Space, Time, and Space-time. Spatial Cognition and Computation 4, 39-68.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.

Goodchild, M. E. (2003). Geographic information science and systems for environmental management. Annual Review of Environment and Resources 28, 493-519.

Goovaerts, P. (1997). Geostatistics for Natural Resources Evaluation. Oxford Univ. Press, New York.

Grenon, P. and Smith, B. (2003). SNAP and SPAN: Towards Dynamic Spatial Ontology. Spatial Cognition & Computation 4, 69-104.

Güting, R. H. and Schneider, M. (2005). *Moving Objects Databases*. Morgan Kaufmann, New York.

Henzinger, T. A. (1996). The Theory of Hybrid Automata. Proceedings of the 11th Symposium on Logic in Computer Science (LICS'96), IEEE.

Hornsby, K. and Egenhofer, M. (2000). Identity-Based Change: A Foundation for Spatio-Temporal Knowledge Representation. International Journal of Geographical Information Science 14, 207-224.

Ihaka, R. and Gentleman, R. (1996). R: A Language for Data Analysis and Graphics. Journal of Computational and Graphical Statistics 5, 299-314.

Krasner, G. E. and Pope, S. T. (1988). A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming 1, 26-49.

Krueger, C. W. (1992). Software reuse. ACM Computing Surveys 24, 131-183.

Martin, D. (2003). Extending the automated zoning procedure to reconcile incompatible zoning systems. International Journal of Geographical Information Science 17, 181-196.

Medak, D. (2001). Lifestyles. In: A. U. Frank, Raper, J., & Cheylan, J.-P. (Ed.) *Life and Motion of Socio-Economic Units. ESF Series.* Taylor & Francis, London.

Meinel, G. and Neubert, M. (2004). A comparison of segmentation programs for high resolution remote sensing data. International Archives of Photogrammetry and Remote Sensing, **XXXV**, 1097-1105.

Meyer, B. (1990). Lessons from the design of the Eiffel libraries. Communications of the ACM 33, 68-88.

Mockus, A., Fielding, R. and Herbsleb, J. (2002). Two case studies of open source software development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology 11.

Navulur, K. (2006). *Multispectral Image Analysis Using the Object-Oriented Paradigm*. CRC Press, Boca Raton, CA.

Openshaw, S. and Alvanides, S. (2001). Designing zoning systems for representation of socio-economic data. In: A. Frank, J. Raper and J. Cheylan (Eds.), *Time and Motion of Socio-Economic Units.* Taylor and Francis, London.

Parks, B. O. (1993). The Need for Integration. In: M. J. Goodchild, B. O. Parks and L. T. Steyaert (Eds.), *Environmental Modelling with GIS.* OUP, Oxford, 31-34.

Pebesma, E. and Bivand, R. (2005). Classes and methods for spatial data in R. R News 5, 9-13.

Pedrosa, B., Câmara, G., Fonseca, F. and Souza, R. C. M. (2002). TerraML - A Cell-Based Modeling Language for an Open-Source GIS Library. II International Conference on Geographical Information Science (GIScience 2002), Boulder, CO, 2002.

Queiroz, G. R. (2003). Algoritmos Geométricos para Bancos de Dados Geográficos: Da Teoria à Prática na TerraLib (Geometric Algorithms for Spatial Databases: From Theory to Practice in TerraLib). Computer Science. São José dos Campos, INPE. **MsC**.

Rodrigues, V. L., Andrade, M. V. A., Queiroz, G. R. and Magalhães, M. (2006). An efficient map overlay algorithm for TerraLib. VIII Brazilian Symposium on GeoInformatics, GeoInfo2006, Campos do Jordão, SP, Brazil, INPE.

Rodrigues, V. L., Cavalier, A. P., Andrade, M. V. A. and Queiroz, G. R. (2005). Exact Algorithms for Map Manipulation in TerraLib. VII Brazilian Symposium on GeoInformatics, GeoInfo2005, Campos do Jordão, SP, Brazil, INPE.

Rowlingson, B. and Diggle, P. (1993). Splancs: spatial point pattern analysis code in S-Plus. Computers and Geosciences 19, 627-655.

Silva, M. P. S., Camara, G., Souza, R. C. M., Valeriano, D. and Escada, M. I. S. (2005). Mining Patterns of Change in Remote Sensing Image Databases. The Fifth IEEE International Conference on Data Mining, New Orleans, Louisiana, USA.

Sistla, A. P., Wolfson, O., Chamberlain, S. and Dao, S. (1997). Modeling and Querying Moving Objects. Proceedings of the Thirteenth International Conference on Data Engineering422-432.

Veldkamp, A. and Fresco, L. (1996). CLUE: A Conceptual Model to Study the Conversion of Land Use and its Effects. Ecological Modeling 85, 253-270.

Vinhas, L., Ferreira, K. R. and Ribeiro, G. (2007). TerraLib Programming Tutorial. São José dos Campos, Brasil, INPE (avaliable on http://www.terralib.org).

Vinhas, L., Queiroz, G. R., Ferreira, K., Câmara, G. and Paiva, J. A. (2002). Generic Programming applied to GIS Algorithms. IV Brazilian Symposium on Geoinformatics, Caxambu, Brazil.

Vinhas, L., Souza, R. C. M. and Câmara, G. (2003). Image Data Handling in Spatial Databases. V Brazilian Symposium on Geoinformatics, Campos do Jordão, Brazil.

Warmerdam, F. (2007). Shapefile C Library V1.2.

Weber, S. (2004). *The Success of Open Source*. Harvard University Press, Cambridge, 75

Worboys, M. (2005). Event-oriented approaches to geographic phenomena. International Journal of Geographic Information Systems 19, 1-28.