Prototyping GIS Application in Functional Programming

Sérgio S. Costa[1], Ana Paula Aguiar[1], Gilberto Câmara[1]

[1]Earth System Science Center (CST) – Instituto Nacional de Pesquisas Espaciais (INPE) - Av. dos Astronautas, 1758 – 12227-001 – São José dos Campos – SP – Brasil

{ scosta, anapaula, gilberto}@dpi.inpe.br

Abstract. *Functional programming languages satisfy the key requirements for specification languages, having expressive semantics and allowing rapid prototyping. Translating formal semantics is direct, and the resulting algebraic structure is extendible. One of the important uses of functional language for GIS is to enable fast and sound development of new applications. In this paper, we present an introduction to the basic features of TerraHS in prototyping GIS application. TerraHS is software developed in Haskell language for GIS application developing that provide basic spatial operations and structures for prototyping novel ideas in GisScience.*

1. Introduction

Developing geographic information systems is a complex enterprise. GIS applications involve data handling, algorithms, spatial data modeling, spatial ontologies and user interfaces. The diversity of data types as networks, fields, objects, time series and spatial-temporal (Figure 1), is an intimidating problem for GIS developer.

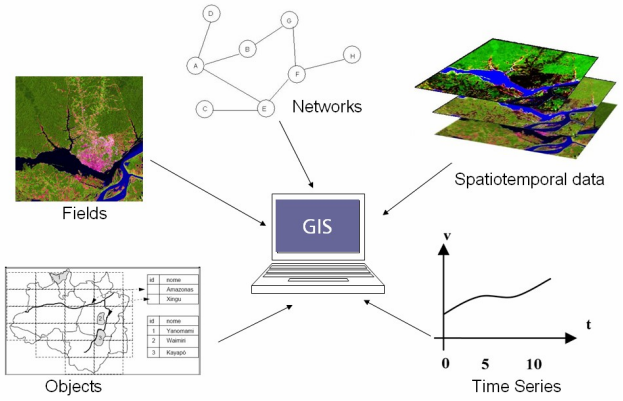


Figure 1 General view of GIS datas

Research in Geographic Information Science has shown than many spatial data can be expressed as algebraic theories [Kuhn 1993; Frank and Kuhn 1995; Frank 1999; Medak 2001]. These algebraic theories formalize spatial components in a rigorous and generic ways. As an answer to the challenges of translation of algebraic specifications into computer languages, there has been a growing interest in functional languages. In this paper, we present an introduction to *TerraHS*, an application development system that enables prototyping novel GisScience ideas in functional language [Costa, Câmara and Palomo 2007].

1. A Brief description of TerraHS

TerraHS links to TerraLib using the *Foreign Function Interface* (FFI) [Chakravarty 2003] and to additional code written in C (`TerraLibC`), which maps the FFI to TerraLib methods. In the Figure 2, lighter colors represent the parts provided by TerraHS and darker colors represent the existing components.

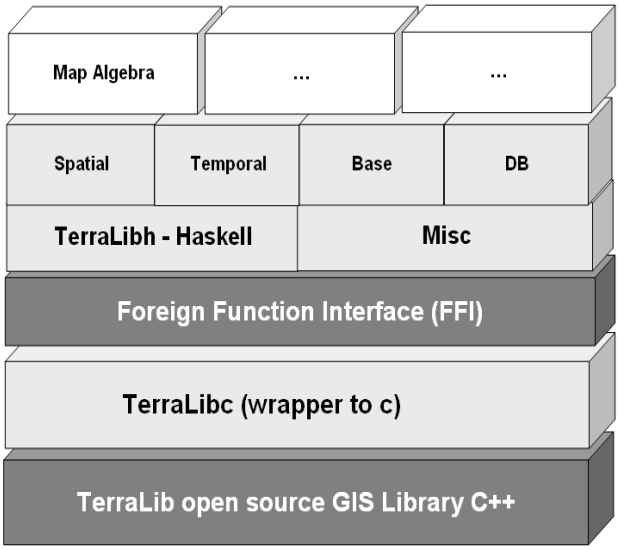


Figure 2. Architecture of TerraHS

Lower layers provide basic services over which upper layer services are implemented. In the bottom layer, TerraLib supports different spatial database management and many spatial algorithms. In the second layer, TerraLibC maps the Terralib C++ methods to the Haskell FFI. In the third layer, the FFI enables calling the TerraLibC functions from Haskell. In the fourth layer, TerraLibH contains the modules that map TerraLib C++ classes to Haskell data types and functions, TeGeometry.hs, TeDatabase.hs and so on. Misc, contains the modules that provide auxiliary functions to TerraHS, such as string and generic functions. In the fifth layer, contains data types and services for supporting to new specific algebras in the last layer. They describe algebraic abstract data types for spatial, temporal, database and base data types. More details and examples are described in http://lucc.ess.inpe.br/doku.php?id=terrahs.

2.1 Installing and using

TerraHS is available in Cabal1 format from HackageDB site2. However, firstly it is necessary to download and to install the following requirements: (a) GHC-6.10.1, (b) MySQL-5.0.41, (c) TerraView-3.2.0 and (e) TerraLibC-0.5.

---

[1] Cabal (http://www.haskell.org/cabal) is a standard way of packaging Haskell source code that makes it easy to build and install

[2] Direct address to download TerraHS: http://hackage.haskell.org/cgi-bin/hackage-scripts/package/terrahs

After that, download terrahslib-0.6.rar and unzip the archive. Thus will create a directory terrahslib-0.6. From a prompt, move into the created directory and run the following commands:

```
runhaskell Setup configure
runhaskell Setup build
runhaskell Setup install
```

Install the Eclipse Software Development Kit (Eclipse SDK) application, obtaining the latest version from the Eclipse site (www.eclipse.org). After that, is necessary to download and install the Haskell plugin via the built-in Update Manager in Eclipse, more details in: http://eclipsefp.sourceforge.net/.

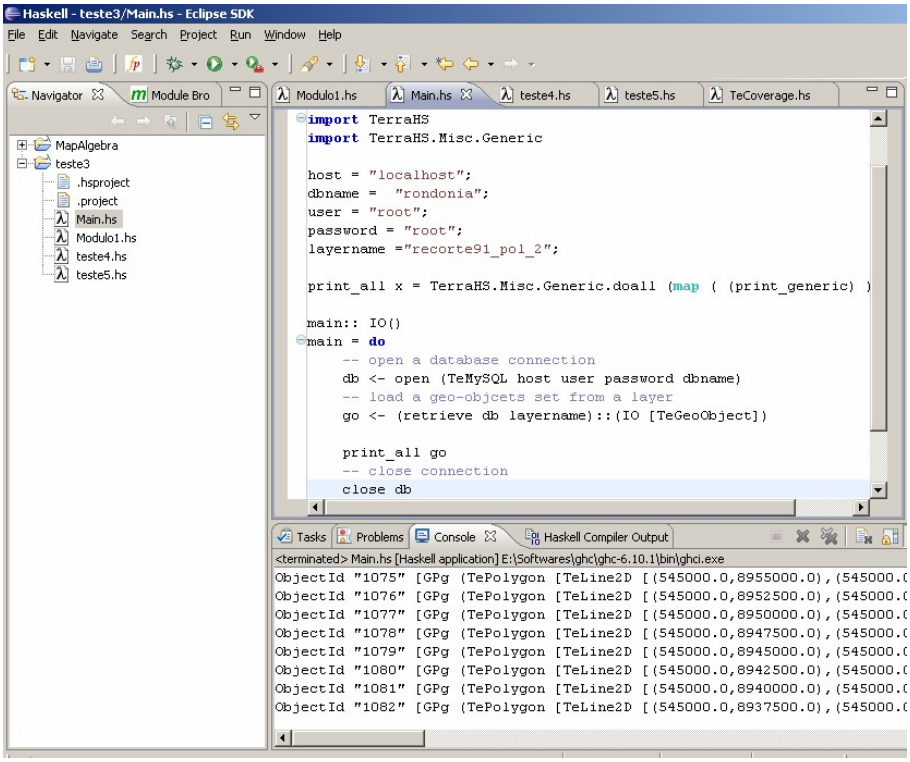
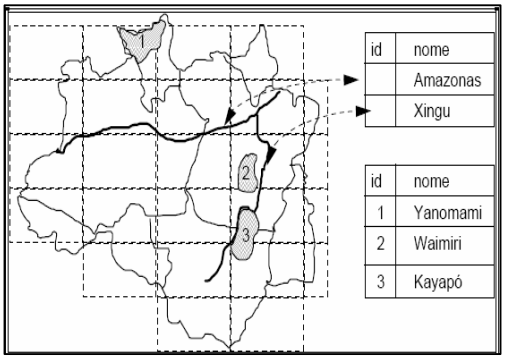


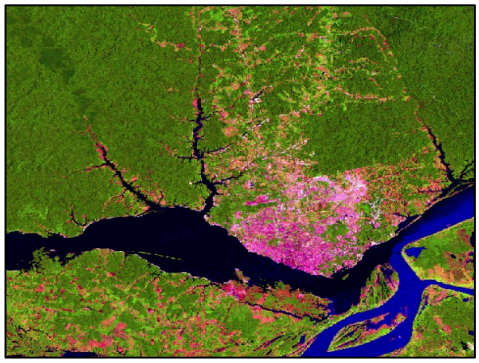
Figure 3. Using TerraHS in Eclipse software

2.3 Examples of reading and writing GIS data from database

In GIS application, there are two major data types, geo-object and geo-field. Geo-objects represent individualizable entities of the geographic realm. They are phenomena that may have one or more graphical representations, which correspond to the geo-referenced set of co-ordinates that describe the object's location. A geographical field or geo-field represents a continuous geographical variable over some region of the Earth [Câmara, Freitas and Casanova 1995].



a. Example of geo object



b. Example of geo fields

Figura 1. Examples of data types supported in TerraHS

A geo-object in TerraHS is a triple:

```
data TeGeoObject = TeGeoObject ObjectId [Attribute] [TeGeometry]
```

ObjectId is used to give to each geo-object a unique identity to distingue a geo-object in TerraLib database. Attributes are the descriptive part of a geo-object. An attribute has a name and a value. *TeGeometry* is the spatial part, which can have different representations, as point, line, polygon and cell.

A geo-field in TerraHS is provided by raster data type:

```
type Grd a = [[a]]
data TeRaster a = TeRaster (Grd a)
```

TeRaster is a generic structure for handling raster data independent from format (eg. float, integer, char), size of each raster element or storage device. In below, is showed an example of a program, which load a raster from database, apply a inverse operation and save to a new raster.

```
main :: IO()
main  = do
   -- open a database connection
   db <- open (TeMySQL "localhost" "root" "pass" "amazon")
   --- load a raster from  database
   rs <- (retrieve db "raster_in")::(IO [TeRaster])
   -- apply a inverse operation
   let rs_inv = lift1 (\x->255 - x) rs
   -- save the new raster
   store db "inv_image" rs_inv
   -- close connection
   close db
```

The structure of a program that loads a geo-object is the same; the single difference is showed below:

```
-- load a geo-objcets set from a layer
go <- (retrieve db "layerin")::(IO [TeGeoObject])
```

In this case, the return of *retrieve* function is a set of *TeGeoobject*.

3. Prototyping GIS applications

The Section 2 shows the two major data type provided by TerraHS software. However, sometimes we need to build the specific data types in functional language. For example, consider the algebra of moving objects proposed by [Güting and Schneider 2005]. They define a basic type moving point (*mpoint*) as a mapping between a temporal reference and a spatial location. A simple implementation of the *mpoint* data type in Haskell is:

```
data UPoint = UPoint(Point,Point,Time)
type MPoint = [UPoint]
```

For this reason, we provided a common approach to deal with specific data types into TerraHS, Figure 4.

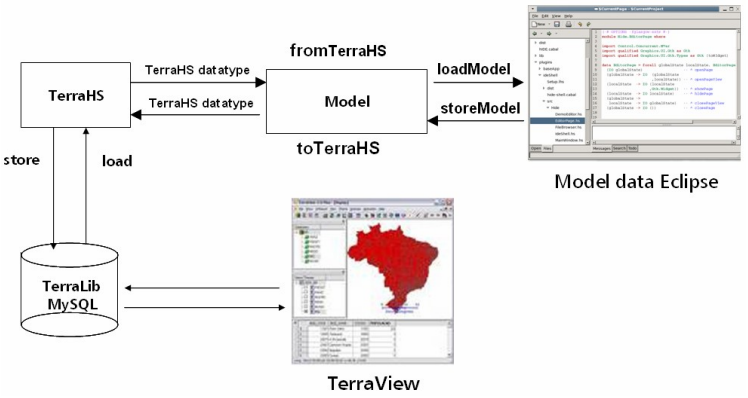


Figure 4. Retrieving and storing specific data type from spatial database

The goal of this approach is to enable a generic and elegantly way to retrieve and store a specific data type from database. In sum, we implement a new module in TerraHS that provide two *type classes,* as showed below.

```
class ModelConvert m te te where
   toTerraHS :: m -> te
   fromTerraHS :: te -> m

class (ModelConvert m te) => ModelPersistence m te where
   storeModel db p m = (store db p (toTerraHS m) )
   loadModel db p = (load db p) >>= ( \go -> return(fromTerraHS go) )
```

The *type class ModelConvert* consists of two operations, *toTerraHs* and *fromTerraHS* that describe how to map between TerraHS data types and specific data types. These operations must be instantiated for specific data type, such as MPoint.

```
instance Model [MPoint] [TeGeoObject] where …
```

The second class provides generic functions for storage and retrieval specific data types from a spatial database. The axioms of *type class ModelPersistence* is generic, it can be applied to different types. We can now write a program that reads and writes the specific data types from a spatial database, as showed below.

```
main :: IO()
main  = do
   -- open a database connection
   db <- open (TeMySQL "localhost" "root" "pass" "amazon")
   --- load a set of "MyType" from  database
   mdata <- (retrieveModel db "layername")::(IO [MPoint])
   -- save in a new layer
   storeModel db "newLayer" mdata
   -- close connection
   close db
```

4. Final Remarks

In this paper, we showed a brief introduction of a latest version of TerraHS software, which include spatial and temporal types. Besides, we present a generic and elegantly way to access and write specific data types from a spatial database, given that the major application of TerraHS is to provide a concise way to test new abstract data types.

References

Câmara, G., U. Freitas and M. Casanova (1995). Fields and Objects Algebras for GIS Operations. III Brazilian Symposium on Geoprocessing, São Paulo, USP.

Chakravarty, M. (2003). "The Haskell 98 foreign function interface 1.0: an addendum to the Haskell 98 report." Retrieved 12/05/2006, 2006, from http://www.cse.unsw.edu.au/~chak/haskell/ffi/.

Costa, S. S., G. Câmara and D. Palomo (2007). TerraHS: Integration of Functional Programming and Spatial Databases for GIS Application Development. Advances in Geoinformatics**:** 127-149 %U http://dx.doi.org/10.1007/978-3-540-73414-7_8.

Frank, A. (1999). One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole. Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'99, Stade, Germany). C. Freksa and D. M. Mark. Stade, Germany, Springer-Verlag**:** 95-107.

Frank, A. and W. Kuhn (1995). Specifying Open GIS with Functional Languages. Advances in Spatial Databases—4th International Symposium, SSD '95, Portland, ME. M. Egenhofer and J. Herring. Berlin, Springer-Verlag. **951:** 184-195.

Güting, R. H. and M. Schneider (2005). Moving objects databases, Morgan Kaufmann Publishers.

Kuhn, W. (1993). Metaphors Create Theories for Users. Spatial Information Theory. A. Frank and I. Campari. Berlin, Springer-Verlag. **716:** 366-376.

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