TerraHS: Programação Funcional e Modelo de Dados Espaço-Temporal

Sergio S. Costa
Instituto Nacional de Pesquisas Espaciais
Divisão de Processamento de Imagens
São José dos Campos – SP – Brasil
scosta@dpi.inpe.br

Gilberto Câmara
Instituto Nacional de Pesquisas Espaciais
Divisão de Processamento de Imagens
São José dos Campos – SP – Brasil
gilberto@dpi.inpe.br

Resumo

Esse artigo discute o uso de programação funcional na prototipação e implementação de modelos de dados espaço-temporais. O trabalho descreve um ambiente chamado TerraHS, que pode ser usado no desenvolvimento desses modelos em linguagem funcional e sua aplicação.

Palavras-chave: *prototipação, haskell, terralib, programação funcional, geoinformática*

1. Introdução

Os ambientes de sistemas de informação geográfica são de uma complexidade cada vez maior. Com a introdução de novas aplicações com serviços baseados em localização e modelos dinâmicos, os desenvolvedores de SIG enfrentam desafios crescentes. Em [4] é argumentado que os SIGs atuais são muito complexos e que é importante a utilização de modelos que possibilitem a construção desses sistemas a partir de pequenas partes. Dentro desse escopo, ele propõe a construção de sistemas geográficos a partir de álgebras. Essas álgebras podem ser construídas em linguagens funcionais, gerando assim especificações que possam ser testadas e validadas.

O objetivo desse artigo é realizar uma discussão preliminar do uso da programação funcional como ferramenta de prototipação, especificação e construção de modelos de dados espaço-temporal. A hipótese desse trabalho é que uso de programação funcional pode auxiliar os cientistas nas definições de novos modelos de dados espaço-temporais. Nesse artigo será desenvolvida parcialmente uma álgebra para objetos móveis [6] em Haskell. Essa álgebra utilizará uma biblioteca (denominada TerraHS) que está ainda em fase inicial de desenvolvimento e também será descrita nesse artigo.

2. Programação Funcional e Especificação Algébrica

Programação funcional é um paradigma de programação onde um programa é composto por um conjunto de funções e todas as operações são executadas por funções que tomam como parâmetros outras funções. As funções são aplicadas recursivamente ou por composição e têm como resultado valores. Alguns exemplos desse paradigma são: Lisp, ML, Scheme, Miranda e Haskell.

O conceito de *especificação algébrica* foi proposto na década de 70 [8] para a definição de interfaces de tipos abstratos de dados. Especificações algébricas definem o comportamento de uma operação a partir de um axioma, permitindo assim descrever o comportamento de tipos de dados independentemente da implementação [5]. Uma especificação algébrica é constituída por três partes: um conjunto de *sorts* (objetos), um conjunto de operações aplicadas a esses *sorts* e um conjunto de axiomas que definem o comportamento desses *sorts*. As linguagens funcionais e as especificações algébricas possuem similaridades em pontos essenciais como o formalismo matemático e expressividade [5]. O exemplo a seguir descreve em Haskell uma pequena álgebra de pontos.

```
class Points p where
      x :: p -> Float
      y :: p -> Float
      equal :: p -> p -> bool
      equal a b = (xa == xb)&&
      (ya == yb)
```

Uma especificação algébrica escrita em Haskell consiste de classes e funções que descrevem axiomas e operações [9]. Funções definidas no nível de classe representam axiomas (`equal`) e funções definidas no nível de instância representam operações (`x` e `y`). No exemplo abaixo as funções que retornam as coordenadas dos pontos (`x` e `y`) foram descritas no nível de instância [9]:

```
data Point = Pt String Float Float
instance Points Point where
      x (Pt name xval yval) = xval
      y (Pt name xval yval) = yval
```

Em [4] é considerado que o uso de álgebras no desenvolvimento de sistemas permite combinar diferentes resultados e que uma linguagem funcional permite este tipo de construção. Em [5] é argumentado que linguagens funcionais apresentam todas as características essenciais para especificação algébrica, com a vantagem de gerarem código executável. Neste mesmo artigo, os autores propõem que uma linguagem funcional (Haskell) poderia ser utilizada como linguagem de especificação para o consórcio OpenGis (`www.opengis.net`). Em [9], os autores realizam uma comparação do uso de um método formal em relação a um método semiformal na especificação de sistemas no domínio espacial. Para esta comparação os autores comparam a UML (*Unified Modelling Language*) com programação funcional (Haskell).

As principais características das linguagens funcionais para o desenvolvimento de sistemas de informação geográficas são: permitir unir especificação e prototipação, permitir construção de complexos sistemas a partir de pequenas partes, utilização de estruturas algébricas na representação de modelo de dados e maior clareza e robustez nas especificações de sistemas computacionais [4].

2.1 Exemplo: Álgebra para Objetos Móveis

Em [1] é argumentado que fenômenos espaciais, como o cadastro urbano, uso e ocupação da terra, fluxos hidrológicos e poluição são inerentemente dinâmicos e as representações estáticas comumente utilizadas não os capturam de forma adequada. Deste modo, é justificado o interesse no desenvolvimento de álgebras que sejam espaço-temporais, ou seja, álgebras que lidam com o espaço, o tempo e ambos simultaneamente.

Um dos trabalhos mais relevantes de uma álgebra espaço-temporal é encontrado em [6], onde o autor apresenta uma álgebra para objetos móveis, ou seja, objetos que se movem continuamente no tempo e espaço. Nesse trabalho o autor define duas abstrações de objetos moveis: pontos móveis (*mpoint*) e regiões móveis (*mregion*). Com estas duas abstrações, mostradas na Figura 1, o autor captura o conceito de pontos e regiões móveis no tempo e espaço.

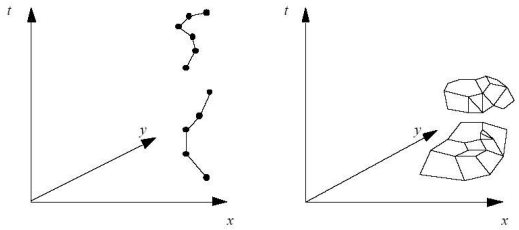


Figura 1: Representação discreta de pontos e regiões móveis. FONTE: [6]

A primeira abstração, `mpoint`, descreve um objeto cuja relevância está nas posições ocupados por esse no espaço ao longo do tempo. A segunda abstração, `mregion`, é utilizada em casos que a área do objeto também é relevante. Um exemplo de `mpoint` é o deslocamento de veiculo ou pessoa, já uma `mregion` poderia ser o movimento de uma tempestade ou queimada numa determinada região. Esse modelo permite responder diversas consultas que envolvem os aspectos espaciais e temporais como:

- *Quais são os vôos partindo de São Paulo que percorrem mais de 5000km?*
- *Quais os pares de aeronaves que durante um vôo estiveram perto até 500m?*

3. TerraHS: Ambiente para Construção de Aplicativos Geográficos.

Considerando o potencial de programação funcional como suporte para construção de sistemas de geoinformação, este artigo descreve a arquitetura do ambiente TerraHS. O propósito desse ambiente é permitir a construção de protótipos de forma mais rápida e com suporte a banco de dados. O TerraHS é uma interface em Haskell (www.haskell.org) para a biblioteca TerraLib (www.terralib.org). Essa interface tem o objetivo de possibilitar que programas escritos nessa linguagem acessem recursos disponíveis na TerraLib.

Para construir o ambiente TerraHS, usamos o Haskell 98 Foreign Function Interface (FFI), que permite executar chamadas de código escrito em chamadas de C [2]. Como a TerraLib foi

implementada em ANSI-C++, não é possível criar uma interface direta entre esses ambientes. Para solucionar esta questão foi criada uma camada intermediária em C que faz acesso aos objetos e funções da TerraLib . Na Figura 2 apresenta a arquitetura do TerraHS, onde as cores mais claras representam o que está em implementação.

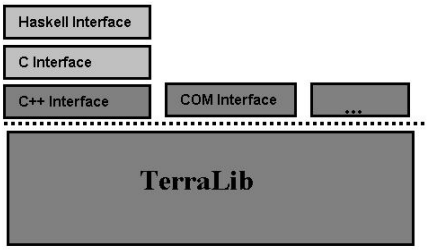


Figura 2: Arquitetura do TerraHS

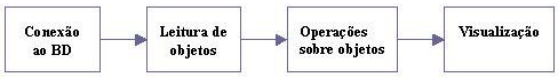
Os principais recursos previstos para esse ambiente estão divididos em três grupos de funcionalidades principais. Na Tabela 1 são mostrados os principais tipos de dados e funções.

Tabela 2: Recursos do TerraHS

Recursos	Tipos	Funções
Banco de Dados	TeDatabase	temysql_connect, loadLayer, loadTable, loadGoSet, selectable, dbname, close, errorMessage
Layer	TeLayer	getLine, electLines, getPoints,selectPoints, getPolygons, selectPolygons, name
Geo-Objetos	TeLine2D TePoint, TePolygon GeoObject	getX, getY , getId, distance, crosses, intersect, disjoint, touch

3.1 Escrevendo um programa em TerraHS

Nessa seção apresenta de forma prática um programa escrito em Haskell que utilize o ambiente TerraHS. Esse programa segue uma estrutura genérica de um software SIG, como pode ser visto na figura abaixo.



A seguir é mostrado a implementação de um programa em TerraHS:

```
main :: IO ()
main = do
       db <- Terrahs.temysql_connect
       host user password dbname
       layer <- Terrahs.loadLayer db
       layername
       lineSet <- Terrahs.getLines layer
       layername
       line1 <- Terrahs.getLine lineSet
       1
       line2 <- Terrahs.getLine lineSet
       2
       cruza <- Terrahs.crosses line
       line2
       print cruza
```

O exemplo acima é um programa bem simples, onde é feito uma conexão a um banco de dados MySQL (`temysql_connect`) , lido duas linhas de um determinado layer (`getLines, getLine`) e verificado o cruzamento das mesmas (`crosses`).

3.2 Experimentos

Nessa seção são mostrados alguns dos experimentos e resultados com a utilização do ambiente proposto, TerraHS.

3.2.1 Objetos Móveis em TerraHS

Como já foi mencionado anteriormente o modelo de objetos móveis [6] é composto por duas abstrações: pontos móveis (`mpoint`) e regiões móveis (`mregion`). Dado que o objetivo desse artigo é uma discussão preliminar foi trabalhada apenas a primeira abstração, ou seja, pontos móveis. Essa álgebra foi dividida em três módulos: temporal, espacial, e espaço-temporal.

Módulo Temporal

Nesse módulo foram definidos os tipos e operações temporais e não se utilizou nenhum recurso do ambiente TerraHS. Os tipos principais implementados foram:

```
data Date = Date (Int,Int,Int)
data Time = Time (Int,Int,Int)
data Instant=Instant(Date,Time)
data Interval = Interval
(Instant,Instant,Bool,Bool)
```

O tipo `Instant` representa um determinado instante que é composto pela data mais o tempo de ocorrência de um fenômeno. `Interval` representa um determinado intervalo temporal que pode ser

compreendido entre dois instantes no caso de intervalo finito. Os dois parâmetros *booleanos* indicam se um determinado intervalo é aberto a direita ou esquerda, indicando assim um intervalo infinito. Algumas operações são:

```
before::Interval->Interval->Bool
after::Interval->Interval->Bool
```

Essas operações verificam a ordem de dois intervalos, permitindo responder se um determinado intervalo antecede ou precede um outro.

Módulo Espacial

O componente TerraHS disponibiliza alguns tipos espaciais próprios, como o `TePoint` e o `TeLine`. Procurando seguir mais fielmente o modelo propostos em [6], nesse trabalho optou-se por não utilizar a estrutura de dados de representação da linha dada pelo TerraHS e sim definir uma estrutura própria, como pode ser visto:

```
data Line =Line [Seg]
```

Para a representação de um ponto utilizou-se o fornecido pelo TerraHS, bastando para isso definir o tipo Point como um tipo derivado do TePoint:

```
type Point = TePoint
```

Para as operações espaciais utilizou-se as operações disponíveis no TerraHS. As principais operações espaciais utilizadas foram: `crosses`, `intersect`, `inside` e `disjoint`.

Modulo Espaço-Temporal

Depois de definido um conjunto de estruturas de dados e operações básicas, definiu-se os tipos principais para lidar com objetos móveis (`MPoint` e `MRegion`). A estrutura de dados `MPoint` é definida como um conjunto de `UPoint` que representa uma unidade temporal. Um `UPoint` é formado por um ponto inicial, um ponto final e um intervalo temporal associado a esta transição:

```
data UPoint = UPoint Point Point
Interval
data MPoint = MPoint [UPoint]
```

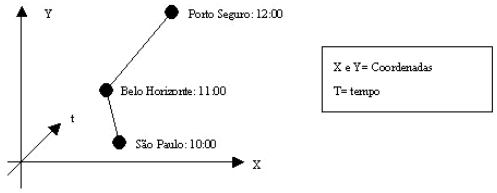
A partir destas definições é criado um conjunto de operações que envolvem os tipos básicos, espaciais, temporais e espaço-temporais:

```
trajectory:: MPoint->Line
deftime:: MPoint->Interval
mduration::MPoint->Duration
```

A operação `trajectory` é definida como a projeção de um ponto móvel em um plano. A função `deftime` retorna o intervalo de tempo compreendido por um ponto móvel. A função `mduration` retorna a duração de um ponto móvel, ou seja, o tempo de existência daquele ponto móvel.

Aplicando a Álgebra de Objetos Móveis

Pontos móveis são capazes de representar movimentos de objetos no tempo e espaço, onde não importa a área do objeto em análise. Um vôo é um exemplo de um ponto móvel. Nos banco de dados atuais os atributos temporais são expressos como colunas de uma tabela. No exemplo da Figura 3, que representa um vôo de São Paulo a Porto Seguro com uma escala em Belo Horizonte, o atributo temporal pose ser representado na tabela de atributo, como é visto na tabela da Figura 3.



Object_id	Cidade	Instante_Hora	Instante_Data
1	São Paulo	10:00	12/10/2005
2	Belo Horizonte	11:00	12/10/2005
3	Porto Seguro	12:00	12/10/2005

Figura 3: Exemplo de uma rota

Através dos dados espaciais e os atributos temporais pode-se construir uma função que receba esses dois parâmetros e gere uma `mpoint` que represente essa "rota". O `mpoint` do exemplo acima é composto por dois `upoints` (São Paulo - Belo Horizonte e Belo Horizonte - Porto Seguro).

Seguindo o exemplo acima foi feito um experimento ainda hipotético. Montou-se uma base de dados que representa o deslocamento de um veículo entre algumas cidades do estado de Minas Gerais. Essa base de dados era composta por três *layers*, onde cada um correspondia o movimento de um determinado veículo. Esses *layers* continham a informação espacial dos pontos de parada desse veículo e a informação temporal que indicava o

instante que aquele veículo parou naquele ponto. Alguns exemplos são:

- Congonhas, Belo Horizonte e Curvelo
- Carandaí, Barbacena e Juiz de Fora

Após a construção da base de dados escreveu-se um programa que acessa esses dados e executa algumas operações, como visto abaixo:

```
main:: IO()
main = do
       db <- Terrahs.temysql_connect
       host user password dbname
       goSet_1 <- Terrahs.loadGoSet db
       "layer_1"
       goSet_2 <- Terrahs.loadGoSet db
       "layer_2"
       mp1 <- convert2MPoint (goSet_1)
       mp2 <- convert2MPoint (goSet_1)
       let line_1 = trajectory mp1
       let line_2 = trajectory mp2
       cruza1 <- Terrahs.crosses
       (linetoteline line_1)
       (linetoteline line_2)
       print cruza1
```

No exemplo acima, fazemos a conexão ao banco, carregamos alguns objetos de dois *layers* distintos, geramos os pontos móveis a partir desses objetos e executamos algumas operações. E interessante ressaltar a facilidade de alternação entre SGBDs, o que implica na alteração de uma única linha que contêm a função de conexão, ou seja, a função `temysql_connect`.

4. Considerações Finais

Esse trabalho descreveu os resultados iniciais no desenvolvimento de um ambiente denominado TerraHS. O TerraHS encontra-se ainda em fase inicial de desenvolvimento. Atualmente o ambiente ainda não contempla todos os recursos existentes na TerraLib. O compromisso atual é definir as principais operações e tipos de dados necessários a um aplicativo geográfico, pois a TerraLib possui uma gama muito vasta de recursos.

Esse ambiente tem como objetivo auxiliar o desenvolvimento de conceitos de geoinformática usando linguagem funcional, dados as vantagens do uso dessa linguagem apontadas no decorrer do texto. Mesmo em fase inicial, foi possível a utilização de algumas operações básicas e isso permitiu a construção e validação de uma álgebra de objetos móveis em programação funcional. Os próximos passos a ser realizados são:

- Concluir o desenvolvimento do ambiente TerraHS
- Descrever uma álgebra espaço-temporal
- Aplicar essa álgebra a um determinado problema

Para a conclusão do desenvolvimento do TerraHS é necessário definir melhor quais são os tipos de dados e funções necessárias a uma aplicação SIG. Para a descrição de uma álgebra espaço-temporal poderá ser utilizada alguns modelos já definidos [3] e [7] ou uma extensão desses. O problema de aplicação ainda está em estudo, porem uma opção é o teste de uma álgebra espaço-temporal que responda algumas questões relacionadas ao PRODES (www.prodes.org).

Agradecimentos

Esse trabalho está tendo o apoio da GISPLAN (www.gisplan.com) e está sendo desenvolvido por um bolsista da empresa.

5. Referências

[1] Câmara, G., Davis, C., Monteiro, A.M., "Banco de Dados Geográficos", 2005.

[2] Chakravarty, M.M.T., "C->Haskell, An Interface Generator for Haskell", 2005.

[3] Erwig, M., Schneider, M., "Spatio-Temporal Predicates", IEEE Transactions on Knowledge and Data Engineering, 14 2002 881-901.

[4] Frank, A. *One Step up the Abstraction Ladder: Combining Algebras - From Functional Pieces to a Whole*. In: Freksa, C., Mark, D. (ed.): COSIT - Conference on Spatial Information Theory. Lecture Notes in Computer Science 1661. Springer-Verlag,1999. 95-108.

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

[6] Güting, R.H., Böhlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N.A., Schneider, M., Vazirgiannis, M., "A Foundation for Representing and Querying Moving Objects", ACM Transactions of Database Systems, 25 2000.

[7] Güting, R.H., Bohlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N., Nardelli, E., Schneider, M., Viqueira, J.R.R. *Spatio-temporal Models and*

Languages: An Approach Based on Data Types. In: Koubarakis, M.(ed.): Spatio-Temporal Databases. Springer, Berlin 2003.

[8] Guttag, J., Horning, J., "The Algebraic Specification of Abstract Data Types", Acta Informatica, 10 1978 27-52.

[9] Winter, S., Nittel, S., "Formal information modelling for standardisation in the spatial domain", International Journal of Geographical Information Science, 17 2003 721--741.