UNIVERSIDADE FEDERAL DE OURO PRETO

DEPARTAMENTO DE COMPUTAÇÃO

INSTITUTO DE CIÊNCIAS EXATAS E BIOLÓGICAS

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

LEANDRO DA SILVA SANTOS

LUATERRALIB – PROTOTIPAGEM RÁPIDA DE APLICAÇÕES GEOGRÁFICAS

Monografia apresentada ao Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Dr. Tiago Garcia de Senna Carneiro

OURO PRETO
2008

LEANDRO DA SILVA SANTOS

LUATERRALIB – PROTOTIPAGEM RÁPIDA DE APLICAÇÕES GEOGRÁFICAS

Monografia apresentada ao Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto, como requisito para a obtenção do grau de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Dr. Tiago Garcia de Senna Carneiro

OURO PRETO
2008

TERMO DE APROVAÇÃO

LEANDRO DA SILVA SANTOS

LUATERRALIB – PROTOTIPAGEM RÁPIDA DE APLICAÇÕES GEOGRÁFICAS

Monografia apresentada ao Departamento de Computação do Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto, como requisito para a obtenção do grau de Bacharel em Ciência da Computação, e aprovada pela Banca Examinadora abaixo assinada:

Ouro Preto, ____ / ____ / _____

_____

Orientador: Prof. Dr. Tiago Garcia de Senna Carneiro

_____

Membro: Prof. Dr. Ricardo de Oliveira Duarte

_____

Membro: Prof. Dr. Frederico Gadelha Guimarães

Resumo

LuaTerraLib é um projeto de software que tem como objetivos tornar eficiente o processo de desenvolvimento de aplicações geográficas, além de tornar mais fácil o ensino e aprendizado desse processo produtivo. Estes objetivos são alcançados através de duas abordagens. A primeira visa o reuso de serviços e padrões já estabelecidos na área do geoprocessamento. A segunda procura atenuar os problemas causados pelo uso de linguagens de programação de uso geral, substituindo-as por linguagens de script. Para avaliar o impacto dessas abordagens no processo produtivo de aplicações geográficas, a biblioteca LuaTerraLib foi desenvolvida de forma a estender a linguagem de programação Lua com objetos e serviços definidos pela biblioteca C++ TerraLib, destinada a manipulação de banco de dados geográficos.

Abstract

LuaTerraLib is a software project which has two goals, to improve the process of GIS application development and to make easier the teaching and learning of this technological process. Two different approaches have been used to reach these goals. The former was the use of already established patterns and services for GIS application development. The later consists on attenuating problems caused by the usage of general purpose programming languages replacing them by high level script languages. To assess this methodology effects, the LuaTerraLib library extends the Lua programming language with types, objects and algorithms provided by the TerraLib GIS C++ library.

Lista de Ilustrações

Lista de Tabelas

Sumário

1 Introdução

Na última década, os conhecimentos e as tecnologias para o processamento de dados geográfico vêm se popularizando impulsionados pela crescente disponibilidade e facilidade de acesso a dados gratuitos capturados por sensores remotos (Crepani & Medeiros, 2005), pela crescente oferta de plataformas de software gratuitas destinada ao desenvolvimento de Sistemas de Informação Geográfica (SIG), e por mudanças culturais que fizeram com que profissionais de diferentes áreas passassem a reconhecer as aplicações SIG como ferramentas eficazes e indispensáveis para melhor compreensão, análise e organização do espaço geográfico (Câmara, Davis, & Monteiro, Introdução a Ciência da Geoinformação, 2001). Contudo, o processo de desenvolvimento de aplicações geográficas continua pouco conhecido na comunidade de desenvolvedores de software devido, principalmente, ao caráter multidisciplinar deste campo científico que mistura conhecimentos da geografia, cartografia e computação. Este último fato dificulta seu ensino em cursos de graduação exclusivos a uma dessas grandes áreas. Apenas um pequeno grupo de instituições de pesquisa ou ensino, além de algumas empresas, domina este processo produtivo.

Neste trabalho é proposta uma metodologia para popularizar conhecimentos e tecnologias produzidas pela comunidade de geoinformática e para tornar mais eficiente o processo de desenvolvimento de aplicações geográficas. Para isso, o projeto LuaTerraLib oferece aos desenvolvedores de software uma plataforma sobre a qual aplicações geográficas podem ser desenvolvidas pelo reuso de padrões e serviços já estabelecidos e pelo uso de uma linguagem de programação interpretada e de alto-nível. Desta forma, a rápida prototipagem de soluções é apoiada e o foco do desenvolvedor pode ficar restrito a questões relativas ao domínio da aplicação em desenvolvimento, ao invés de questões relacionadas às complexas estruturas sintáticas e semânticas de uma linguagem de programação de uso geral como, por exemplo, a linguagem C++.

A biblioteca C++ LuaTerraLib registra no ambiente de execução da linguagem Lua (Ierusalimschy, Figueiredo, & Celes, 2007) todos os tipos, objetos e algoritmos que formam a API (*Application Programming Interface)* de serviços públicos da biblioteca C++ TerraLib (Câmara, et al., 2000), que fornece serviços para armazenamento, recuperação e análise de dados geográficos em sistemas gerenciadores de bancos de dados relacionais (SGBD).

LuaTerraLib é estruturada conforme o padrão de projeto de software *wrapper* (Gamma, Helm, Johnson, & Vlissides, 2004) e utiliza-se do projeto tolua++ (Manzur & Celes, 2006) para estender a linguagem Lua.

Nas seções abaixo do texto será expresso o arcabouço teórico que dá sustentação ao projeto, a metodologia utilizada para a implementação do mesmo, os resultados alcançados e a conclusão que irá se referir ao que foi proposto e ao que foi alcançado.

2 Fundamentação Teórica

Esta seção do texto apresenta todo o arcabouço teórico necessário para o desenvolvimento do projeto LuaTerraLib, serão expostos conceitos referentes aos projetos: TerraLib, Lua, padrão de projeto *wrapper* e a ferramenta tolua++. O projeto TerraLib oferece um conjunto de serviços e definições para se tratar entidades geográficas num SIG. O projeto Lua é uma linguagem de scripts livre que têm a oferecer um conjunto de características pertinentes a este grupo de linguagens como o fato de ser uma linguagem interpretada. O padrão de projeto *wrapper* é uma forma de estabelecer a comunicação entre objetos que possuem diferentes interfaces, neste caso a TerraLib e a linguagem Lua. A ferramenta tolua++ é um *software* para a produção de *wrapper* de C++ para Lua.

2.1 A biblioteca TerraLib

A biblioteca TerraLib é um projeto de *software* livre (Vinhas & Ferreira, 2005) desenvolvida pelo INPE que vem atender as necessidades da comunidade de desenvolvimento de aplicações geográficas. A TerraLib é uma biblioteca escrita em C++ de código fonte aberto (Vinhas & Ferreira, 2005) e foi elaborada para servir como base para os desenvolvedores de aplicações SIG (Sistemas de Informação Geográfica).

A TerraLib é uma biblioteca que fornece funções para a decodificação de dados geográficos, estrutura de dados espaço-temporais e algoritmos de análise espacial (Câmara, Neves, Monteiro, Souza, Paiva, & Vinhas, 2002). A TerraLib também propõe um novo modelo para uma base de dados geográfica (Câmara, Neves, Monteiro, Souza, Paiva, & Vinhas, 2002).

A arquitetura da biblioteca TerraLib pode ser dividida em duas partes: o *Kernel* e os *Drivers* (Vinhas & Ferreira, 2005). O *Kernel*, figura 2.1, da TerraLib é composto por estruturas de dados espaço-temporais, suporte a projeção cartográfica, operadores espaciais, uma interface para o armazenamento de dados espaço-temporais em bancos de dados objeto-relacionais e um mecanismo de visualização (Vinhas & Ferreira, 2005).

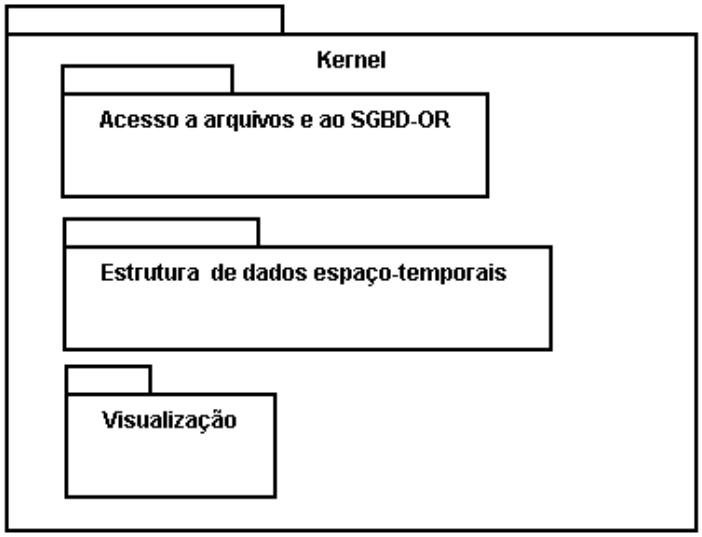

Figura 2.1 – Arquitetura do *Kernel* da biblioteca TerraLib.

Os *Drivers* da TerraLib fornecem a implementação da interface de armazenamento e recuperação dos dados geográficos (Vinhas & Ferreira, 2005). Eles também são responsáveis por fornecer a decodificação de dados geográficos para formatos abertos (ASCII-SPRING) e proprietários (Shapefiles) (Vinhas & Ferreira, 2005).

A TerraLib, figura 2.2, ainda pode ser vista como uma camada de comunicação entre a aplicação geográfica e a base de dados.

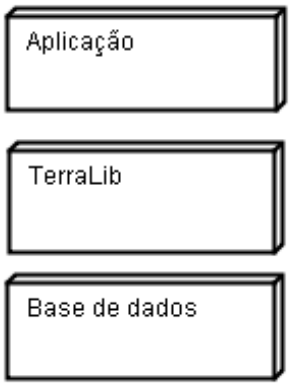

Figura 2.2 – A camada de acesso TerraLib.

2.1.1 Modelo conceitual

A TerraLib além de propor um modelo de armazenamento de dados em um SGBD-OR (Sistema Gerenciador de Bancos de Dados – Objeto Relacional) disponibiliza um modelo conceitual de banco de dados geográficos sobre o qual elabora seus algoritmos de

processamento (Vinhas & Ferreira, 2005). Abaixo segue a exemplificação das entidades do modelo conceitual de bases de dados geográficas da TerraLib (Vinhas & Ferreira, 2005):

- **Banco de Dados**: é uma entidade que pode ser vista como apenas um repositório da informação geográfica que tem como requisitos básicos a capacidade de ser acessível a um software, conter uma extensão capaz de criar tipos abstratos espaciais ou ainda ser capaz de armazenar campos binários longos.
- **Layer**: é uma entidade que tem a função de agregar um conjunto de informações sobre uma determinada área geográfica e que compartilham um conjunto de atributos.
- **Representação**: é uma entidade que trata do modelo de representação da componente espacial dos dados do *layer* e pode ser vetorial ou matricial. Na representação vetorial temos pontos, linhas, áreas e outras estruturas mais complexas formadas a partir dessas. Na matricial há o suporte a representação de grades regulares multidimensionais. Esta entidade ainda permite que um geo-objeto seja representado por diferentes representações vetoriais.
- **Projeção Cartográfica**: é a entidade responsável por representar a referência geográfica da componente espacial do dado geográfico. As projeções cartográficas são responsáveis por permitirem colocar a superfície da Terra sobre o plano minimizando as distorções neste processo. A TerraLib ainda fornece um conjunto de projeções padrão para o desenvolvedor.
- **Tema**: é a entidade que permite selecionar o dado através de critérios sobre seus atributos descritivos ou sobre sua componente espacial. Ele ainda fornece a apresentação visual do dado.
- **Vista**: é a entidade na qual se reúne um conjunto de temas que estarão sobre a mesma projeção e serão visualizados conjuntamente.
- **Visual**: é a entidade que agrega as informações referentes à forma como as primitivas geométricas irão ser visualizadas _ cor, espessura e etc.
- **Legenda**: é a entidade que coloca um conjunto de dados sobre um mesmo visual.

2.1.2 Classes do modelo conceitual

A TerraLib é uma biblioteca não dependente de SGBD porque se utiliza de uma interface para manipulação de uma base de dados (Vinhas & Ferreira, 2005). A interface para manipulação do SGBD é fornecida pela classe TeDatabase, figura 2.3, que é implementa por cada *driver* da TerraLib.

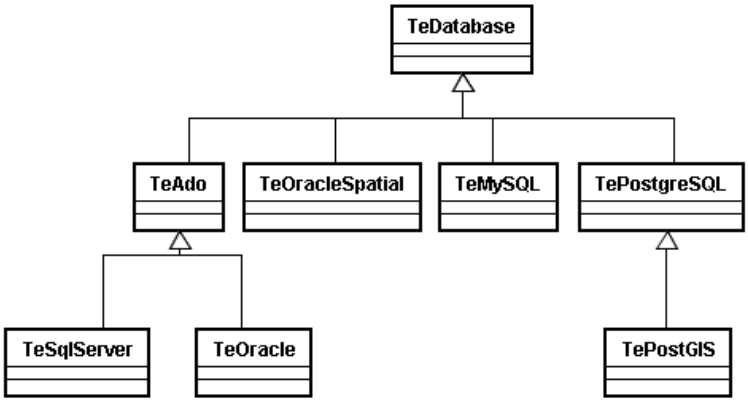

Figura 2.3 – *Drives* implementados pela biblioteca TerraLib.

Na figura temos como ocorre o relacionamento entre as principais classes que dão suporte ao modelo conceitual da TerraLib.

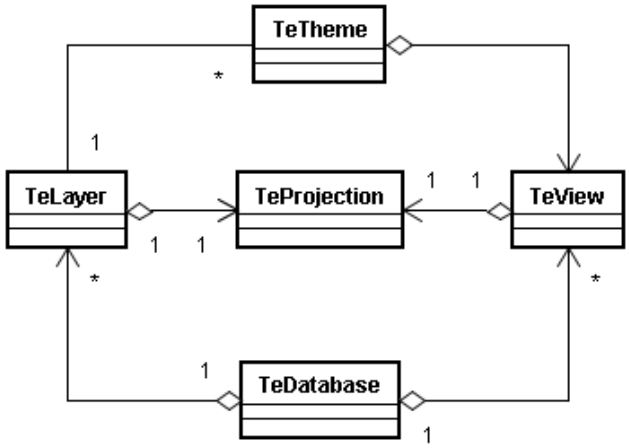

Figura 2.4 – Principais classe do modelo conceitual da TerraLib.

2.1.3 Modelo de geometrias

Todas as geometrias da TerraLib são derivadas da classe TeGeometry (Vinhas & Ferreira, 2005). Cada geometria contém um identificador único, uma referência ao seu menor retângulo envolvente e à sua projeção e um identificador do objeto ao qual está associada (Vinhas & Ferreira, 2005). As geometrias bi-dimensionais da TerraLib são elaboradas através da classe TeCoord2D (Vinhas & Ferreira, 2005) e são:

- **Ponto**: é implementada como uma única instância da classe TeCoord2D sendo representada pela classe TePoint.
- **Linha**: é composta por um ou mais segmentos sendo representa pela classe TeLine2D. A linha é implementada como um vetor de uma ou mais instâncias da classe TeCoord2D.
- **Anel**: é uma linha fechada (o primeiro ponto é igual ao último), ela é representada pela classe TeLinearRing e é implementada como uma única instância da classe TeLine2D salvo os requisitos.
- **Polígono**: expressa a delimitação de uma área que pode conter vazios. O vazio é expresso como um polígono filho. O polígono é representado pela classe TePolygon e é implementado com um vetor da classe TeLinearRing no qual o primeiro elemento representa o pai e os outros os filhos.

2.2 A linguagem Lua

Lua é uma linguagem de script livre (Ierusalimschy, Figueiredo, & Celes, 2007) desenvolvida pela Tecgraf que reúne elementos que favorecem a produtividade no processo de desenvolvimento de software. Hoje, Lua é uma linguagem reconhecida internacionalmente e usada em larga escala, como exemplo, podemos citar a indústria de desenvolvimento de jogos dentre outras.

2.2.1 Principais características

A linguagem de programação Lua possui uma sintaxe simples (figura 2.5) a qual é formada por um pequeno número de construções sintáticas que permitem seu rápido aprendizado (Hirschi, 2007). A linguagem de programação Lua ainda possui mecanismos como tabela associativa e meta-tabela que permitem o desenvolvimento de aplicações segundo o paradigma da OO (Ierusalimschy, Figueiredo, & Celes, 2007).

```
function fact(n)
   if n == 0 then
      return 1
   else
      return n * fact(n - 2)
   end
end
```

Figura 2.5 – Exemplo de código em Lua.

A linguagem de programação Lua, por ser interpretada, possibilita alterações no código para o teste de novas soluções sem a necessidade de uma nova compilação do sistema. Para fins de depuração, o código da aplicação pode ser alterado mesmo durante sua execução. Portanto, protótipos passam a ser produzidos de maneira mais eficiente.

Lua é uma linguagem de grande portabilidade, pois, é completamente compatível com o padrão ANSI/ISO C, portanto, é portável para as plataformas Unix e Windows.

2.2.2 Comparação a outras linguagens de scripts

Quando comparada a outras linguagens de scripts como Python e Ruby (tabela 2.1), Lua apresentou vantagens relacionadas ao desempenho das aplicações visto que o código interpretado por Lua se mostrou mais eficiente em termos de velocidade de execução (Mascarenhas & Ierusalimschy, 2008). O projeto Lua ainda apresenta uma estrutura mais compacta (Ierusalimschy, Figueiredo, & Celes, 2007).

Tabela 2.1 – Quadro comparativo entre as linguagens Lua, Python e Ruby.

Características Linguagem	Multi- Thread	Reparação de Erro	Coletor de Lixo	Orientada a Objetos	Dinamicamente Tipada	Multi- Plataforma
Lua	X		X		X	X
Python	X			X		X
Ruby	X	X		X		X

Frente a linguagens de uso geral como C++, o projeto Lua possui vantagens relacionadas ao fato de ser interpretada. Aplicações podem ser rapidamente desenvolvidas sem exigir a compilação de seus módulos. O código fonte em Lua pode ser alterado em tempo de execução, ou mesmo, durante sua depuração, o que contribui para a rápida prototipagem de soluções. Como contrapartida, Lua apresenta um desempenho pior que C++. Contudo, no desenvolvimento de aplicações de alto desempenho, módulos que possuem esse requisito podem ser desenvolvidos em C e ter a API registrada em Lua (Ierusalimschy, Figueiredo, & Celes, 2007).

2.3 O Padrão de projeto *Wrapper*

O *Wrapper* (também conhecido como *Adapter*) é um padrão de projeto de software (Gamma, Helm, Johnson, & Vlissides, 2004) classificado como um padrão estrutural. Possui o objetivo de converter a interface de uma classe em outra interface, esperada pelos clientes. O *Wrapper* é necessário quando possuímos dois objetos que precisam se comunicar, mas as suas interfaces são diferentes. Neste caso, os objetos em questão são os sistemas TerraLib e a linguagem de programação Lua.

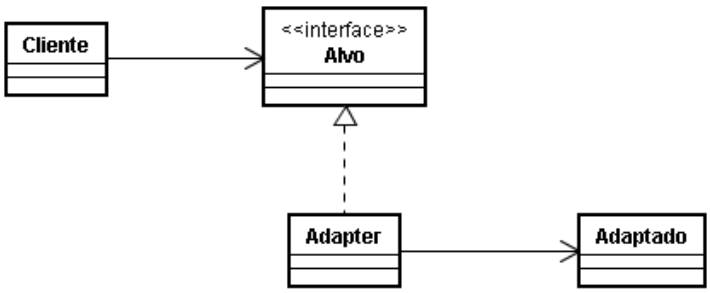

Figura 2.6 – Diagrama de Classe para o padrão *Wrapper*.

Um diagrama de classes descrevendo a estrutura do *Wrapper* pode ser visto na figura 2.6. Nesta figura, definimos quatro entidades: o Alvo, que é a interface que você quer usar; o Cliente, que é o objeto que irá interagir com a interface alvo; O Adaptado, que é a interface que precisa ser adaptada para a interface Alvo e o Adapter, que "transforma" a interface do Adaptado na interface Alvo.

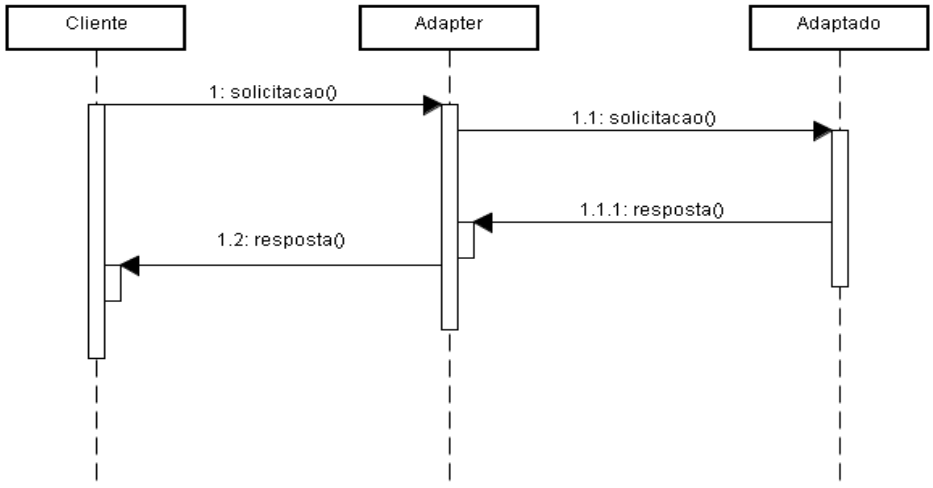

Figura 2.7 – Diagrama de seqüência para o padrão *Wrapper*.

O funcionamento do *Wrapper* é simples. O Cliente precisa comunicar-se com o Adaptado, mas possuem interfaces incompatíveis. Então, ele comunica-se com o Adapter,

fazendo a este uma solicitação. O Adapter repassa a solicitação para o Adaptado. O Adaptado precisa então devolver uma resposta para o Cliente, mas novamente não é possível devido à incompatibilidade de interfaces. A solução é enviar a resposta para o Adapter que irá em seguida encaminhá-la para o Cliente, completando assim o ciclo de comunicação (Freeman & Freeman, 2004). Esse funcionamento pode ser traduzido em um diagrama de seqüências como o mostrado na figura 2.7. No caso específico desse trabalho, o cliente é a linguagem de programação Lua, o Adaptado é um objeto TerraLib escrito em C++, e o alvo é uma interface comum que será proposta neste trabalho para solucionar o problema de comunicação entre TerraLib e a linguagem de programação Lua.

2.4 A ferramenta tolua++

A tolua++ é uma ferramenta para integração de C/C++ com a linguagem de programação Lua. A tolua++ utiliza a API de Lua e um conjunto de outros métodos para mapear constante, variáveis externas, funções, métodos e classes de C/C++ para Lua (Manzur & Celes, 2006).

A ferramenta tolua++ tem como entrada um arquivo contendo a interface a qual se quer exporta para Lua (Manzur & Celes, 2006). Como saída, ela produz um *wrapper* que irá conter a interface exportada.

As constantes exportadas por tolua++ são definidas pelas clausuras: define ou enum de C/C++ (figura 2.8). A exportação se dá através da criação de variáveis globais com nomes e valores correspondentes às de C/C++ (Manzur & Celes, 2006).

```
#define NAME [ VALUE ]

enum {
  NAME1 [ = VALUE1 ] ,
  NAME2 [ = VALUE2 ] ,
  ...
  NAMEn [ = VALUEn ]
};
```

Figura 2.8 – Clausuras define e enum de C/C++.

As variáveis externas exportadas por tolua++ são definidas por um qualificador extern (figura 2.9). A tolua++ define essas variáveis em Lua como variáveis globais, desta forma, poderá se acessar a variável normalmente (Manzur & Celes, 2006).

```
[extern] type var;
```

Figura 2.9 – Clausura extern de C/C++.

As funções exportadas por tolua++ têm que ter uma assinatura correta na gramática de C/C++. A tolua++ ainda permite a sobrecarga de funções como também a definição de valores padrões para os parâmetros da função (figura 2.10).

```
type funcname (type1 par1[, type2 par2[,...typeN parN]]);

void func (void* p);
void func (Object1* ptr);
void func (Object2* prt);

type funcname (..., typeN-1 parN-1 [= valueN-1], typeN parN [= valueN]);

void func (int a[5]=0);
```

Figura 2.10 – Assinatura de funções em C/C++.

Estruturas definidas em C/C++ pelo qualificador typedf struct (figura 2.11) são suportados plenamente por tolua++. Assim o usuário de Lua pode criar a estrutura e acessar suas variáveis e métodos.

```
typedef struct [name] {
   type1 fieldname1;
   type2 fieldname2;
   ...
   typeN fieldnameN;
} typename;
```

```
typedef struct {
   int x[10];
   int y[10];
} Example;
```

Figura 2.11 – Estrutura em C/C++

Classes e o conceito de herança de C++ são suportados por tolua++. Com relação aos qualificadores de visibilidade tolua++ mantém o seu comportamento em Lua (Manzur & Celes, 2006).

Também, vale ressaltar, a presença de um dispositivo para se utilizar os conceitos de Templates de C/C++ em Lua (Manzur & Celes, 2006). A tolua++ permite ainda a sobrecarga dos seguintes operadores de Lua:

```
operator+   operator-   operator*   operator/
operator<   operator>=  operator==  operator[]
```

3 Materiais e Métodos

Esta seção do texto apresenta os sistemas de computação e a metodologia utilizada para o desenvolvimento da biblioteca C++ e sua implementação na forma de uma camada de software integrado a linguagem de programação Lua. Esta nova camada será doravante chamada LuaTerraLib.

3.1 Sistemas de computação utilizados

Para a elaboração deste projeto foram utilizados os seguintes sistemas de computação:

- TerraLib – biblioteca livre para o desenvolvimento de Sistemas de Informação Geográfica.
- Lua – linguagem de script livre, interpretada e procedural.
- tolua++ - ferramenta para geração de *wrapper* de C++ para Lua.

3.2 O Processo de desenvolvimento de software

A metodologia de desenvolvimento de *software* utilizada foi o modelo em espiral com prototipação de releases e versões. Assim, as etapas de concepção, projeto, implementação, teste e documentação serão executadas de forma cíclica, e a cada ciclo uma nova versão documentada da LuaTerraLib será gerada.

3.3 Requisitos do sistema

O projeto LuaTerraLib deve prover ao ambiente de programação Lua acesso a API pública de serviços geográficos da TerraLib. Ao usuário de Lua será permitido criar e invocar os objetos e métodos da TerraLib. A LuaTerraLib deve permitir ao usuário de Lua exprimir e usar os conceitos relacionados ao paradigma OO aplicados na TerraLib. Desta forma a LuaTerraLib fornece os seguintes serviços:

- funções para a decodificação de dados geográficos para formatos livres e proprietários.

- estrutura de dados espaço-temporais.
- algoritmos de análise espacial.
- suporte a projeção cartográfica.
- operadores espaciais.
- armazenamento e recuperação dos dados em SGBD-OR .

A LuaTerraLib também é portável para qualquer sistema que tenha um compilador compatível com o padrão ANSI/ISO C (plataformas Unix e Windows).

3.4 Arquitetura do sistema

A figura 3.1 mostra a arquitetura em camadas de LuaTerraLib. Na primeira camada, TerraLib fornece todos os serviços disponíveis ao desenvolvimento de aplicações geográficas. Sobre essa camada, LuaTerraLib registra esses serviços na máquina virtual Lua. Na última camada, encontram-se as aplicações geográficas desenvolvidas em Lua.

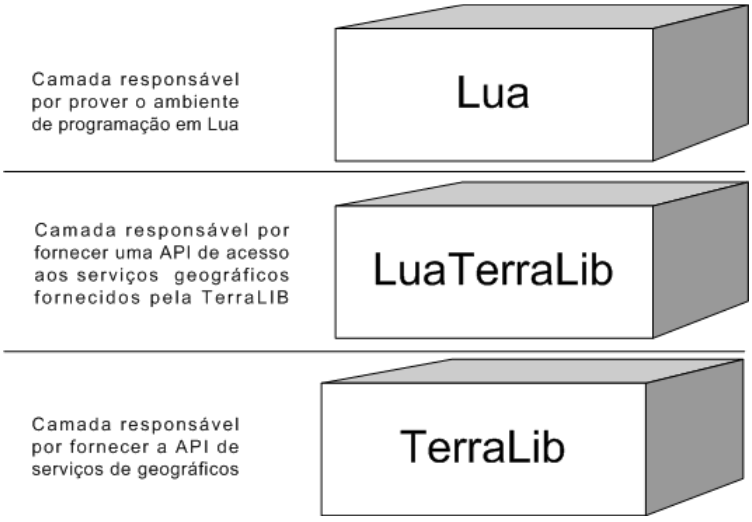

Figura 3.1 – Arquitetura em camadas do projeto LuaTerraLib.

A LuaTerraLib foi desenvolvida como um *wrapper* (Gamma, Helm, Johnson, & Vlissides, 2004) entre a linguagem Lua e a biblioteca TerraLib, como mostra a figura 3.2. Ela converte a interface de uma classe TerraLib em outra interface, que é esperada pelo ambiente de execução Lua.

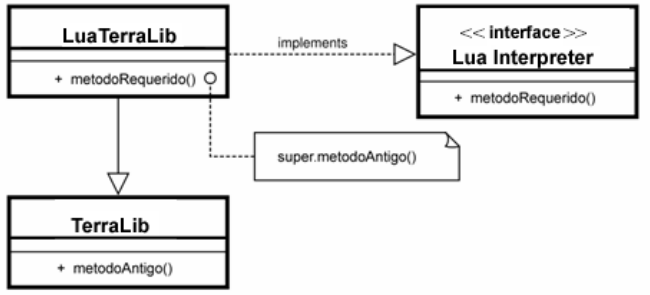

Figura 3.2 – LuaTerraLib é estruturada como um *wrapper* entre Lua e TerraLib.

Na figura 3.3 podemos observar o projeto LuaTerraLib dividido em outros módulos, como se segue abaixo:

- Geometria vetorial – este pacote é responsável por exportar para o ambiente de programação Lua os conceitos e serviços implementados pela TerraLib no tratamento do dado vetorial. A linha e o polígono podem ser citados como exemplos de um dado vetorial.
- Raster – este pacote é responsável por exportar para o ambiente de programação Lua os conceitos e serviços implementados pela TerraLib no tratamento do dado Raster. Como exemplo de dado Raster pode-se citar imagens de sensoriamento remoto.
- Metadado – este pacote é responsável por exportar para o ambiente de programação Lua as classes que implementam a estrutura do banco de dados do tipo TerraLib. Como estruturas características de um banco de dados TerraLib temos, como exemplos, o layer e o tema.
- Armazenamento e recuperação do dado geográfico em SGBD-OR – este pacote é responsável por exportar para o ambiente de programação Lua as classes que implementam a interface de comunicação com o SGBD-OR.
- Elemento espaço-temporal – este pacote é responsável por exportar para o ambiente de programação Lua as classes que implementam os conceitos referentes ao dado espaço-temporal.
- Algoritmo geométrico – este pacote é responsável por exportar para o ambiente de programação Lua as operações topológicas, de conjunto, métricas e outras definidas na TerraLib.
- Funções – este pacote é responsável por exportar para o ambiente de programação Lua um conjunto de funções auxiliares referente à exportação e importação de

arquivos livres ou proprietários para uma base de dados do tipo TerraLib, dentre outras funções.

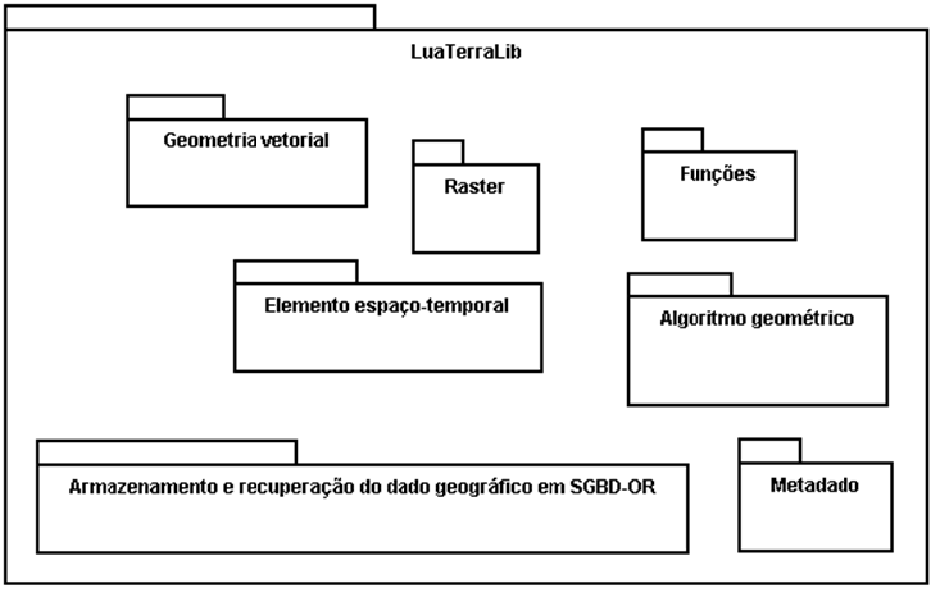

Figura 3.3 – Diagramas de pacotes do projeto LuaTerraLib.

3.5 Especificação do sistema

Esta seção do texto apresenta a especificação de cada módulo da LuaTerraLib: Geometria vetorial, Raster, Metadado, Armazenamento e recuperação do dado geográfico em SGBD-OR , Driver, Elemento espaço-temporal, Algoritmo geométrico e Funções. Será exemplificado nestes itens a hierarquia de classes e seus métodos principais como a função que desempenha.

3.5.1 Geometria vetorial

Este módulo é responsável por exportar os conceitos relacionados a geometrias vetoriais da TerraLib para o ambiente de programação Lua. Desta forma temos que o modelo de geometria da LuaTerraLib é análogo ao da biblioteca TerraLib.

As classes abaixo, exportadas para o ambiente de programação Lua, são originárias da interface pública da TerraLib sendo que todos os objetos necessários ao seu uso foram exportados para Lua. Abaixo, segue as principais classes referentes a geometria vetorial implementadas pela LuaTerraLib:

- TeGeometry – classe responsável por exportar a noção de geometria da TerraLib para o ambiente de programação Lua. Todas as geometrias da LuaTerraLib são filhas de TeGeometry. A classe TeGeometry tem como componentes básicos: um identificador do tipo de geometria e o retângulo envolvente da mesma. Na figura 3.4 estão os principais métodos implementados pela LuaTerraLib.

	TeGeometry () *Empty constructor.*
	TeGeometry (const TeGeometry &other) *Copy Constructor.*
virtual	~TeGeometry () *Destructor.*
void	setBox (const TeBox &box) *Sets the bounding box for the object.*
const TeBox &	box () const *Returns the constant bounding box.*
TeBox &	box () *Returns the bounding box.*
int	geomId () const *Returns the geometry id.*
void	geomId (int id) *Sets the geometryId.*
virtual string	objectId () const *Returns the object unique identification.*
virtual void	objectId (const string &id) *Sets the objectId.*
virtual unsigned int	size () const *Return the geometry size.*
ostream &	operator<< (ostream &os) *Outputs the geometical identification to an output stream.*
virtual bool	isRing () const *Returns TRUE if a geometry is a closed ring.*
virtual TeGeomRep	elemType () *Returns the basic geometry type in a set of geometries structure.*

Figura 3.4 – Lista de membros da classe TeGeometry.

- TeVector – classe responsável por exportar as geometrias que podem ser tratadas com vetores da TerraLib para o ambiente de programação Lua. Na figura temos a interface desta classe.
- TePoint – classe responsável por exportar a representação de ponto 2D da TerraLib para o ambiente de programação Lua. Na figura 3.5 temos um resumo da sua hierarquia de classes deste objeto.

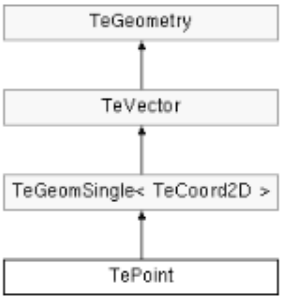

Figura 3.5 – Hierarquia de classes do objeto TePoint.

- TeLine2D – classe responsável por exportar a representação de linha 2D da TerraLib para o ambiente de programação Lua. Na figura 3.6 temos um resumo da sua hierarquia de classes deste objeto.

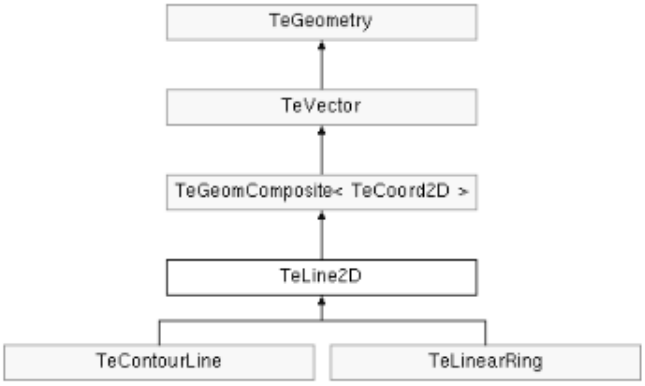

Figura 3.6 – Hierarquia de classes do objeto TeLine2D.

- TeLinearRing – classe responsável por exportar a representação de linha fechada 2D (linha na qual o primeiro ponto é igual ao último) da TerraLib para o ambiente de programação Lua. Na figura 3.7 temos um resumo da sua hierarquia de classes deste objeto.

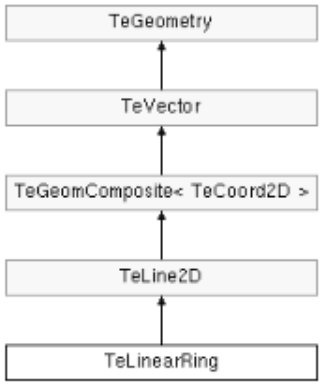

Figura 3.7 – Hierarquia de classes do objeto TeLinearRing.

- TePolygon – classe responsável por exportar a noção de polígono da TerraLib para o ambiente de programação Lua. O TePolygon é implementado como sendo composto por um anel exterior e uma lista de anéis interiores a este. Na figura 3.8 temos um resumo da sua hierarquia de classes deste objeto.

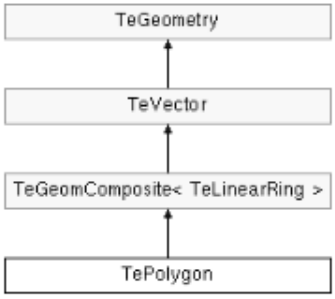

Figura 3.8 – Hierarquia de classes do objeto TePolygon.

- TeCell – classe responsável por exportar a noção de célula da TerraLib para o ambiente de programação Lua. Na figura 3.9 temos um resumo da sua hierarquia de classes deste objeto.

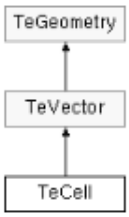

Figura 3.9 – Hierarquia de classes do objeto TeCell.

- TeArc – classe responsável por exportar a noção de arco composto de nós de origem e destino da TerraLib para o ambiente de programação Lua. Na figura 3.10 temos um resumo da sua hierarquia de classes deste objeto.

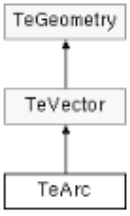

Figura 3.10 – Hierarquia de classes do objeto TeArc.

- TeNode – classe responsável por exportar a noção de nó em uma rede da TerraLib para o ambiente de programação Lua. Na figura 3.11 temos um resumo da sua hierarquia de classes deste objeto.

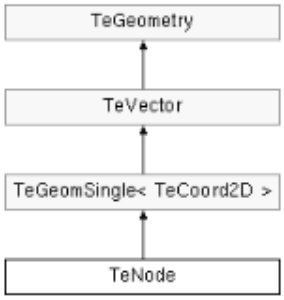

Figura 3.11 – Hierarquia de classes do objeto TeNode.

- TeSample – classe responsável por exportar a noção de ponto 2D associado a um valor qualquer da TerraLib para o ambiente de programação Lua. Na figura 3.12 temos um resumo da sua hierarquia de classes deste objeto.

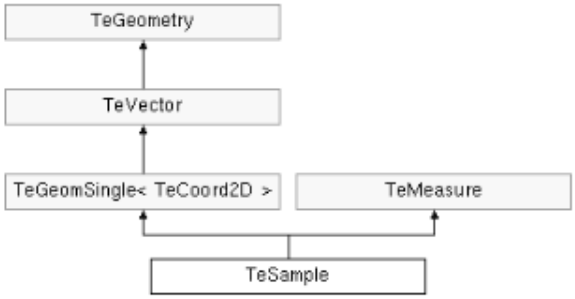

Figura 3.12 – Hierarquia de classes do objeto TeSample.

- TeContourLine – classe responsável por exportar a noção de linha de contorno da TerraLib para o ambiente de programação Lua. Na figura 3.13 temos um resumo da sua hierarquia de classes deste objeto.

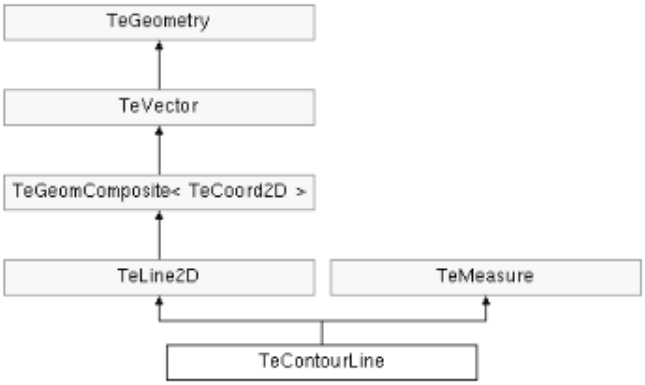

Figura 3.13 – Hierarquia de classes do objeto TeContourLine.

- TeText – classe responsável por exportar a noção de texto da TerraLib para o ambiente de programação Lua. Na figura 3.14 temos um resumo da sua hierarquia de classes deste objeto.

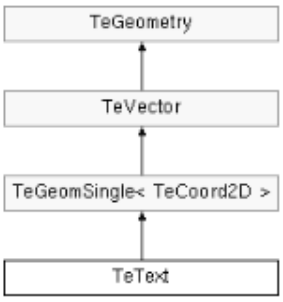

Figura 3.14 – Hierarquia de classes do objeto TeText.

- TePolygonSet – classe responsável por exportar a noção de conjunto de polígonos da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.15 temos um resumo da sua hierarquia de classes deste objeto.

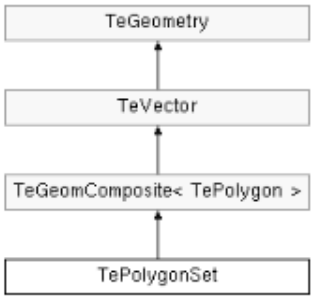

Figura 3.15 – Hierarquia de classes do objeto TePolygonSet.

- TeLineSet – classe responsável por exportar a noção de conjunto de linhas 2D da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.16 temos um resumo da sua hierarquia de classes deste objeto.

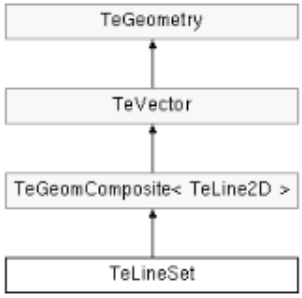

Figura 3.16 – Hierarquia de classes do objeto TeLineSet.

- TePointSet – classe responsável por exportar a noção de conjunto de pontos 2D da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.17 temos um resumo da sua hierarquia de classes deste objeto.

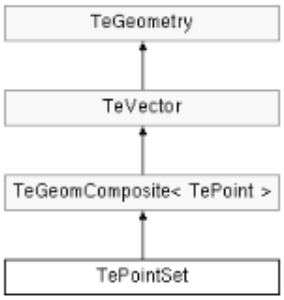

Figura 3.17 – Hierarquia de classes do objeto TePointSet.

- TeCellSet – classe responsável por exportar a noção de conjunto de células da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.18 temos um resumo da sua hierarquia de classes deste objeto.

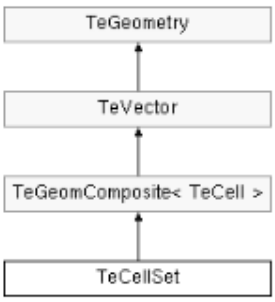

Figura 3.18 – Hierarquia de classes do objeto TeCellSet.

- TeArcSet – classe responsável por exportar a noção de conjunto de arcos da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.19 temos um resumo da sua hierarquia de classes deste objeto.

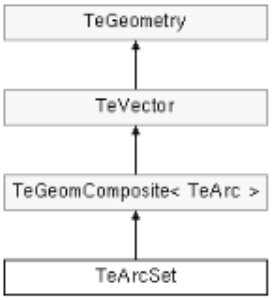

Figura 3.19 – Hierarquia de classes do objeto TeArcSet.

- TeNodeSet – classe responsável por exportar a noção de um conjunto de nós de uma rede da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.20 temos um resumo da sua hierarquia de classes deste objeto.

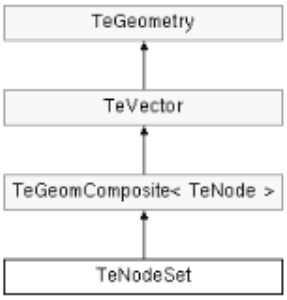

Figura 3.20 – Hierarquia de classes do objeto TeNodeSet.

- TeSampleSet – classe responsável por exportar a noção de conjunto de pontos 2D associados a um valor qualquer da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.21 temos um resumo da sua hierarquia de classes deste objeto.

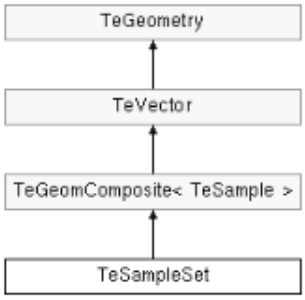

Figura 3.21 – Hierarquia de classes do objeto TeSampleSet.

- TeContourLineSet – classe responsável por exportar a noção de conjunto de linhas de contorno da TerraLib para o ambiente de programação Lua através do padrão de

projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.22 temos um resumo da sua hierarquia de classes deste objeto.

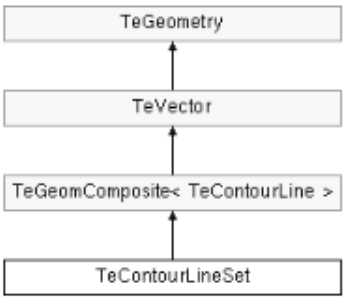

Figura 3.22 – Hierarquia de classes do objeto TeContourLineSet.

- TeTextSet – classe responsável por exportar a noção de conjunto de textos da TerraLib para o ambiente de programação Lua através do padrão de projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.23 temos um resumo da sua hierarquia de classes deste objeto.

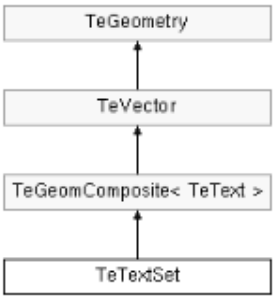

Figura 3.23 – Hierarquia de classes do objeto TeTextSet.

3.5.2 Raster

Este módulo é responsável por exportar os conceitos relacionados ao dado Raster (um tipo de geometria) da TerraLib para o ambiente de programação Lua. As classes abaixo, exportadas para o ambiente de programação Lua, são originárias da interface pública da TerraLib sendo que todos os objetos necessários ao seu uso foram exportados para Lua. Abaixo, segue as principais classes referentes a este módulo que são implementadas pela LuaTerraLib:

- TeRasterParams – classe responsável por exportar o tratamento dos parâmetros do dado raster da TerraLib para o ambiente de programação Lua através do padrão de

projeto Composite (Gamma, Helm, Johnson, & Vlissides, 2004). Na figura 3.24 temos a interface pública deste objeto.

enum	TeRasterInterLeavingMode *Pixels interleaving mode: used by rasters in memory and files in raw formats.*
enum	TeRasterPhotometricInterpretation *Photometric interpretation of a raster data.*
enum	TeRasterStatus *Status of a raster data, in terms of reading and writting possibilities.*
enum	TeRasterCompressionMode *Compression modes of a raster data.*
enum	TeRasterTilingType *Tiling type modes to store raster data in a TerraLib database.*
int	nlines_ *number of lines*
int	ncols_ *number of columns*
double	resx_ *horizontal resolution*
double	resy_ *vertical resolution*
bool	swap_ *a flag to indicate that the values of the elements of the raster are swapped*
bool	useDummy_ *a flag to indicate that raster has dummy values*
char	mode_ *a character indicating the access mode to the raster data: 'r', 'w' or 'c'*
string	decoderIdentifier_ *decoder associated to his raster*
TeRasterInterLeavingMode	interleaving_ *interleaving mode*
string	errorMessage_ *String that contains any error or warning message that raster manipulation might have detected.*
	TeRasterParams () *Default contructor.*
	TeRasterParams (const TeRasterParams &other) *Copy constructor.*
TeRasterParams &	operator= (const TeRasterParams &rhs) *Operator=.*
virtual	~TeRasterParams () *Destructor.*
void	nBands (int n) *Sets the number of bands, or dimentions in a raster data.*
void	projection (TeProjection *proj) *Sets the projection.*
TeProjection *	projection () *Returns the projection.*
TeCoord2D	coord2Index (TeCoord2D &pt) *Transform a coordinate from world domain to line/column domain.*
TeCoord2D	index2Coord (TeCoord2D &pt) *Transform a coordinate from line/column domain to world domain.*
const string &	decName () const *Returns the identifier of the decoder associated to the raster.*
int	nBands () const *Returns the number of bands of the raster.*
void	writeParametersFile () *Saves the parameters in a ASCII File, in TerraLib format.*
void	readParametersFile () *Reads the parameters described in a ASCII File, in TerraLib format.*

Figura 3.24 – Interface da classe TeRasterParams.

- TeRaster – classe responsável por exportar um conjunto de serviços genéricos ao dado raster da TerraLib para o ambiente de programação Lua. Na figura 3.25 temos a interface pública deste objeto.

	TeRaster (const string &filename, const char &mode= 'r') *Constructor from file.*
	TeRaster (TeRasterParams &pars) *Constructor from parameters.*
	TeRaster (int ncols, int nlines, int nbands, TeDataType elemType) *Contructor from commom parameters.*
	TeRaster () *Empty constructor.*
	~TeRaster () *Destructor.*
TeRasterParams &	params () *Returns the parameters of the raster.*
int	nBands () *Returns the number of lines of the raster.*
void	updateParams (TeRasterParams &par) *Update the parametes of a raster file.*
TeGeomRep	elemType () *Returns the type of the geometry.*
void	setDecoder (TeDecoder *dec) *Associate a decoder to a raster.*
bool	status () *Returns status of the raster as a Boolean value.*
TeDecoder *	decoder () *Returns a pointer to the decoder associated to this raster.*
TeProjection *	projection () *Returns a pointer to the raster projection.*
bool	setElement (int col, int lin, double val, int band=0) *Sets the value of a element of the raster.*
bool	getElement (int col, int lin, double &val, int band=0) *Gets the value of a element of the raster.*
bool	fillRaster (TeRaster *dstRaster, TeRasterTransform *transf=0, bool bestRes=true) *Fills a destination raster with the raster elements.*
bool	init () *Initalize the raster decoding tool from its parameters.*
bool	init (TeRasterParams &pars) *Initalize the raster decoding tool from a raster parameters structure.*
void	clear () *Clear internal structures and disable the raster decoding tool.*
TeCoord2D	index2Coord (TeCoord2D pt) *Transform a coordinate from line/column domain to projection domain.*
TeCoord2D	coord2Index (TeCoord2D pt) *Transform a coordinate from projection domain domain to line/column.*
iterator	begin () *Returns an iterator to the first element of the raster.*
iteratorPoly	begin (TePolygon &poly, TeStrategicIterator st, int band=0) *Returns an iterator to the first element of the raster IN or OUT the polygon.*
iterator	end () *Returns the end past one position of the elements of the raster.*
iteratorPoly	end (TePolygon &poly, TeStrategicIterator st, int band=0) *Returns an iterator to the end element of the raster.*
bool	selectBlocks (TeBox &bb, int resFac, TeRasterParams &parBlock) *Select all blocks of raster, in a certain resolution factor that intercepts a given bounding box.*
int	numberOfSelectedBlocks () *Returns the number of blocks selected by the last block selection.*
bool	fetchRasterBlock (TeDecoderMemory *memDec) *Returns the current block of a set selected by the last block selection.*
void	clearBlockSelection () *Clears the current selection of a set selected by the last block selection.*
bool	setElement (int col, int lin, double Rval, double Gval, double Bval, unsigned int=255) *An optimizated method to set values raster images.*
const string &	errorMessage () const *Give access to the last message detected by the raster manipulation.*

Figura 3.25 – Interface da classe TeRaster.

- TeDecoder – classe responsável por exportar os serviços referentes à decodificação do dado raster da TerraLib para o ambiente de programação Lua. Na figura 3.26 temos a interface pública deste objeto.

	TeDecoder () *Empty constructor.*
	TeDecoder (const TeRasterParams &par) *Constructor from raster parameters.*
virtual	~TeDecoder () *Virtual destructor.*
TeRasterParams &	params () *Returns the raster parameters.*
void	updateParams (TeRasterParams &par) *Updates the raster parameters.*
virtual bool	setElement (int col, int lin, double val, int band=0)=0 *Sets the value of a specific raster element.*
virtual bool	setElementRGB (int col, int lin, double Rval, double Gval, double Bval, unsigned int=255) *An optimizated method to set values raster images.*
virtual bool	getElement (int col, int lin, double &val, int band=0)=0 *Gets an specific element (col, lin, band) of a raster data.*
virtual TeCoord2D	coord2Index (TeCoord2D &pt) *Transforms a geographical coordinate to an index (lin, col) coordinate.*
virtual TeCoord2D	index2Coord (TeCoord2D &pt) *Transforms an index (lin, col) coordinate to a geographical coordinate.*
virtual void	init (TeRasterParams &par) *Initializes the internal structures of the decoder from a raster parameters structure.*
virtual void	init ()=0 *Initializes the internal structures of the decoder.*
virtual bool	clear ()=0 *Clears its internal structures.*

Figura 3.26 – Interface da classe TeDecoder.

O relacionamento existente entre as classes TeRasterParams, TeRaster, TeDecoder da LuaTerraLib pode ser expresso pela figura 3.27 abaixo:

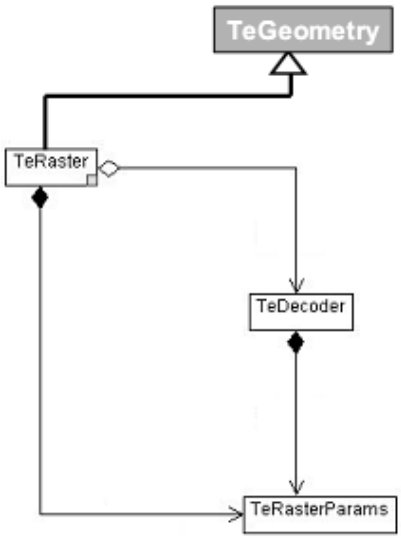

Figura 3.27 – Relacionamento entre as principais classes do módulo Raster.

A LuaTerraLib ainda implementada as seguintes classes de decodificadores da TerraLib que têm como classe base o objeto TeDecoder:

- TeDecoderASCIIGrid – implementa um decodificador para o formato de dado ASCII grid do ESRI.
- TeDecoderJPEG – implementa um decodificador para o formato de imagem JPEG.

- TeDecoderMemory – implementa um decodificador de dado raster armazenado como uma matriz multidimensional na memória.
- TeDecoderMemoryMap – – implementa um decodificador de dado raster em um arquivo binário utilizando as funcionalidades de mapeamento da memória do sistema operacional.
- TeDecoderMrSID – implementa um driver para decodificar imagens no formato MrSID.
- TeDecoderSmartMem – implementa um decodificador de dado raster armazenado como uma matriz multidimensional na memória. O TeDecoderSmartMem é herdado pelas seguintes classes: TeDecoderPAM (decodificador para imagens nos formatos PBM, PGM, ou PPM) e TeDecoderSPR (decodificador para o formato ASCII-SPRING). Na figura 3.28 temos um resumo de sua hierarquia de classes.

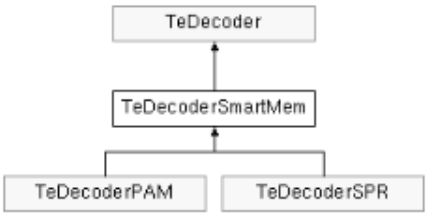

Figura 3.28 – Hierarquia de classes do objeto TeDecoderSmartMem.

- TeDecoderTIFF – implementa um decodificador para dado raster no formato TIFF ou GEOTIFF.
- TeDecoderVirtualMemory – implementa uma estratégia para a decodificação na memorial virtual do dado raster em blocos.

3.5.3 Metadado

Este módulo é responsável por exportar os serviços dos metadados relacionados a uma base de dado da TerraLib para o ambiente de programação Lua. As classes abaixo, exportadas para o ambiente de programação Lua, são originárias da interface pública da TerraLib sendo que todos os objetos necessários ao seu uso foram exportados para Lua. Abaixo, segue as principais classes referentes a este módulo que são implementadas pela LuaTerraLib:

- TeLayer – implementa a exportação dos serviços de manipulação de uma camada numa base de dados do tipo TerraLib para o ambiente de programação Lua.

- TeTheme – implementa a exportação dos serviços de manipulação de um tema numa base de dados do tipo TerraLib para o ambiente de programação Lua. Na figura 3.29 temos a hierarquia de classe deste objeto.

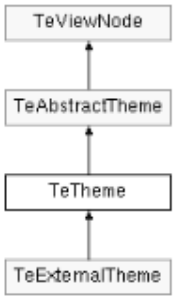

Figura 3.29 – Hierarquia de classes do objeto TeTheme.

- TeView – implementa a exportação dos serviços de agregação e temas que serão processados sobre a mesma projeção para o ambiente de programação Lua.
- TeProjection – implementa a exportação dos serviços de manipulação de uma projeção cartográfica para o ambiente de programação Lua. Na figura 3.30 temos a hierarquia de classe deste objeto.

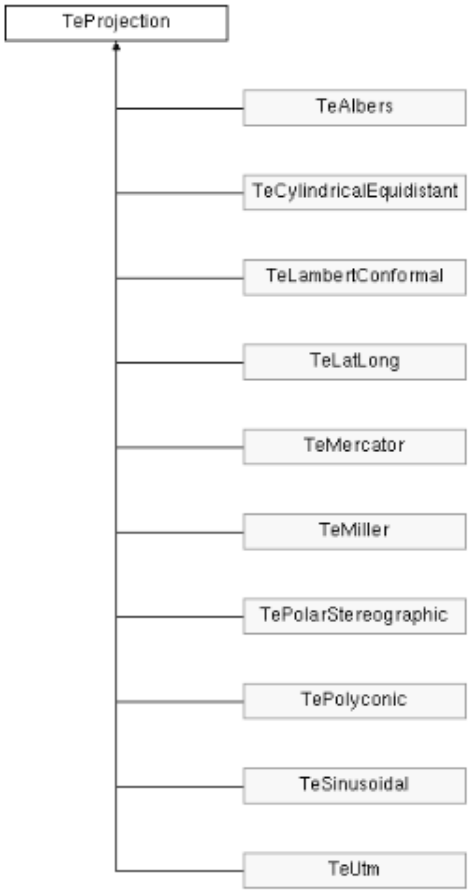

Figura 3.30 – Hierarquia de classes do objeto TeProjection.

O relacionamento existente entre as classes acima da LuaTerraLib pode ser expresso pela figura 3.31 abaixo:

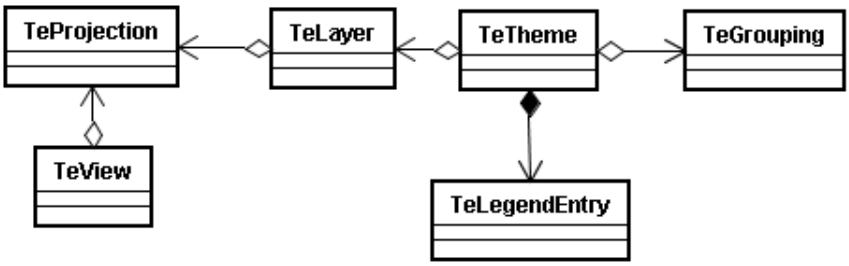

Figura 3.31 – Relacionamento entre as principais classes do módulo Metadado.

3.5.4 Armazenamento e recuperação do dado geográfico em SGBD-OR

Este módulo é responsável por exportar os serviços relacionados ao armazenamento e recuperação da informação geográfica num SGBD-OR da TerraLib para o ambiente de programação Lua. As classes abaixo, exportadas para o ambiente de programação Lua, são originárias da interface pública da TerraLib sendo que todos os objetos necessários ao seu uso foram exportados para Lua. Abaixo, segue as principais classes referentes a este módulo que são implementadas pela LuaTerraLib:

- TeDatabase – implementa a exportação dos serviços de manipulação de um banco de dado do Tipo TerraLib para o ambiente de programação Lua. Na figura 3.32 temos a hierarquia de classe que define os *drivers* disponíveis para o ambiente de programação Lua.

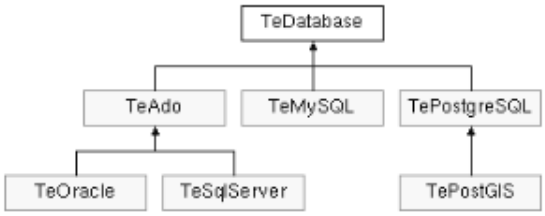

Figura 3.32 – Hierarquia de classe dos *drivers* da LuaTerraLib.

- TeDatabasePortal – implementa a exportação dos serviços referentes a consultas SQL numa base de dados do tipo TerraLib para o ambiente de programação Lua. Na figura 3.33 temos a hierarquia de classe que define as classes para consultas SQL nos diversos drivers disponíveis para o ambiente de programação Lua.

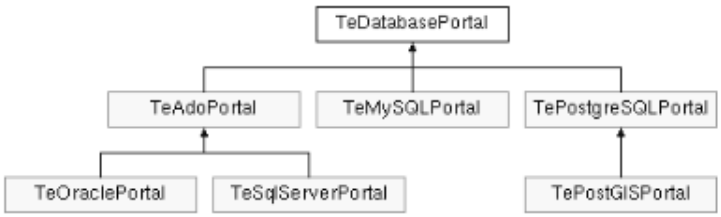

Figura 3.33 – Hierarquia de classe para consultas SQL dos drivers da LuaTerraLib.

3.5.5 Elemento espaço-temporal

Este módulo é responsável por exportar os conceitos relacionados ao dado espaço-temporal definido na TerraLib para o ambiente de programação Lua. As classes abaixo, exportadas para o ambiente de programação Lua, são originárias da interface pública da TerraLib sendo que todos os objetos necessários ao seu uso foram exportados para Lua. Abaixo, segue as principais classes referentes a este módulo que são implementadas pela LuaTerraLib:

- TeSTInstance – implementa a exportação dos serviços que representam uma instância temporal de um elemento ou objeto geográfico da TerraLib para o ambiente de programação de Lua. Na figura 3.34 podemos observar a hierarquia de classes deste objeto.

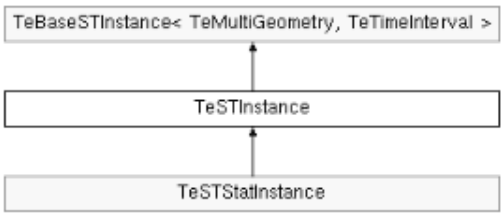

Figura 3.34 – Hierarquia de classes do objeto TeSTInstance.

- TeSTElementSet – implementa a exportação dos serviços que representam todas as instâncias temporais de todos elementos espaciais pertencentes a um tema da TerraLib para o ambiente de programação de Lua. Na figura 3.35 podemos observar a hierarquia de classes deste objeto.

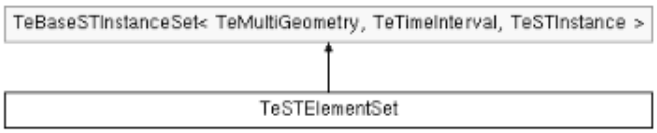

Figura 3.35– Hierarquia de classes do objeto TeSTElementSet.

- TeQuerier – implementa a exportação dos serviços que recuperam instâncias temporais a partir de diferentes fontes (banco TerraLib ou shapefile) da TerraLib para o ambiente de programação de Lua. Na figura 3.36 podemos observar a interface pública desta classe.

	TeQuerier () *Empty constructor.*
	TeQuerier (TeQuerierParams &params) *Constructor from a set of parameters.*
	TeQuerier (const TeQuerier &other) *Copy constructor.*
TeQuerier &	operator= (const TeQuerier &other) *Operator =.*
	~TeQuerier () *Destructor.*
TeTheme *	theme () *Returns the base theme.*
bool	loadInstances (int frame=-1) *Loads the ST instances. If frame > -1, loads only the instances of the frame-th time frame.*
bool	fetchInstance (TeSTInstance &sto) *Gets the current ST instance and moves to the next one. Returns if there is a next instance.*
int	getNumTimeFrames () *Returns the number of time frames generated by a specific chronon.*
bool	getTSEntry (TeTSEntry &tsEntry, int frame) *Gets a temporal serie entry associated to a time frame.*
bool	getTS (TeTemporalSeries &ts) *Gets the temporal series.*
TeTSParams &	getTSParams () *Returns the temporal series parameters.*
TeAttributeList	getAttrList () *Gets the attribute list of the instances.*
TeBox	getBox () *Gets the minimal bounding box.*
int	numElemInstances () *Returns the number of instances loaded by the method "loadInstances".*
TeQuerierParams &	params () *Returns the querier parameters.*
void	clear () *Clear querier structures.*
void	refresh (TeQuerierParams &params) *Refreshes the querier based on a new querier parameters.*
bool	loadGeometries (TeMultiGeometry &geometries, unsigned int &index) *Loads all geometries of the index-th geometry representation.*
bool	loadGeometries (TeMultiGeometry &geometries) *Loads all geometries.*

Figura 3.36 – Interface pública da classe TeQuerier da LuaTerraLib.

4 Resultados

LuaTerraLib fornece ao desenvolvedor Lua acesso a todos os tipos, objetos e algoritmos providos por TerraLib. O código apresentado na figura 4.1 ilustra como aplicações TerraLib são criadas em Lua, especificamente, como criar um novo plano de informação geográfica em um banco de dados armazenado no SGBD MySQL. Na figura 4.2, um novo código importa um arquivo no formato *shape* (*.shp). Na figura 4.3, o código demonstra a conversão de uma coordenada geográfica para outro sistema de coordenadas geográficas Na figura 4.4 é ilustrado o processo de criação de uma base de dados do tipo MySQL. Na figura 4.5 é mostrado como se copia uma layer dentro da base de dados TerraLib. Na figura 4.6 é demonstrada a criação de um tema através da escolha dos elementos e da definição de um novo visual para estes.

```
require 'luaterralib'

-- Datatabase server parameters
 host = "localhost"
 dbname = "teste"
 user = "root"
 password = ""

-- Opens a connection to a database accessible though ADO
db = TeMySQL:new()
if (db:connect(host, user, password, dbname)==false)
then
        print( "Error: ",db:errorMessage())
        return
end

print( "Connection successful to the database",dbname," on MySQL server ",host)

-- Creates a projection
mDatum = TeDatumFactory:make("SAD69")
pUTM = TeUtm:new(mDatum,0.0)

-- Create a new layer called "TesteLayer"
 layerName = "TesteLayer"
if (db:layerExist(layerName))
then
        print( "There is already a layer named TesteLayer in the TerraLib database!")
        db:close()
        return
end

layer =  TeLayer:new(layerName, db, pUTM)

-- Close database
db:close()
return
```

Figura 4.1 – Código em Lua para a criação de um no layer.

```
require 'luaterralib'

host = "localhost";
bdnome = "teste";
user = "root";
password = "";
nlayer = "teste1";

file = "EstadosBrasil.shp";
table = "BrasilIBGE";
row = "SPRROTULO";

bd = TeMySQL:new();
if(bd:connect(host,user,password,bdnome) == false) then
        print("Erro:",bd:errorMessage());
        return;
end

sad69 = TeDatumFactory:make("SAD69");
proj = TePolyconic:new(sad69, -54.0*TeCDR);
layer = TeLayer:new(nlayer, bd, proj);

if TeImportShape( layer, file, table, row,60) then
        print("Numero de poligonos: " ,layer:nGeometries(TePOLYGONS));
        print("Numero de linhas: ", layer:nGeometries(TeLINES));
        print("Numero de pontos: ", layer:nGeometries(TePOINTS));
else
        bd:close();
        return;
end
```

Figura 4.2 – Código em Lua para importar um arquivo *shape (*.shp)*.

```
require 'luaterralib'

dSAD69 = TeDatumFactory:make("SAD69")          -- SAD69 Spheroid

dWGS84 = TeDatumFactory:make("WGS84")          -- WGS84 Spheroid

pUTM = TeUtm:new(dSAD69,-45.0*TeCDR)             -- Origin latitude of -45:0
-- TeCDR means "Convert to Degrees from Radians"

pPolyconic = TePolyconic:new(dWGS84,-45.0*TeCDR)  -- Origin latitude of -45:0
-- TeCDR means "Converte Degrees from Radians"
ptl = TeCoord2D:new(340033.47, 7391306.21)           -- Original coordinate in UTM

-- Conversion from the UTM to the Polyconic projection
pUTM:setDestinationProjection(pPolyconic)

ll = pUTM:PC2LL(ptl)                          -- Convert to Lat Long
pt2 = pPolyconic:LL2PC(ll)          -- Convert to output projection

print("UTM : Polyconic \n")
print(ptl:x(), ptl:y())
print(pt2:x(), pt2:y())

-- Conversion from the Polyconic to the UTM projection
pPolyconic:setDestinationProjection(pUTM)
ll = pPolyconic:PC2LL(pt2)
ptl = pUTM:LL2PC(ll)

print("\nPolyconic : UTM \n")
print(pt2:x(), pt2:y())
print(ptl:x(), ptl:y())

return
```

Figura 4.3 – Código em Lua para conversão de coordenadas.

```
require 'luaterralib'

-- Datatabase server parameters
host = "localhost"
dbname = "teste"
user = "root"
password = ""

-- Creates a new database
db = TeMySQL:new()
if (db:newDatabase(dbname, user, password, host) == false)
then
        print("Error: ", db:errorMessage())
        return
end

print("was created successfully in the MySQL server located in", host)
print("for the user named ", user)

-- Close database
db:close();
```

Figura 4.4 – Código em Lua para criação de uma base de dados.

```
require 'luaterralib'

-- Datatabase server parameters
host = "localhost"
dbname = "teste"
user = "root"
password = ""

-- Open a connection to a MySQL database
db = TeMySQL:new()
if (db:connect(host, user, password, dbname)==false)
then
     print("Error: ",db:errorMessage())
     return
end
print("Database",dbname,"connected on MySQL localhost server!")

-- Load the layer Distritos by its name
layer1 =  TeLayer:new("TesteLayer")
if (db:loadLayer(layer1)==false)
then
     print("The layer Distritos could not be loaded from the database!")
     db:close()
     return
end
print("Layer", layer1:name(), "retrieved!")
-- Show the layer projection
proj1 = layer1:projection()
print("Projection/Datum of the layer Distritos: " ,proj1:name())
print(proj1:datum():name())

-- Create a different projection
sad69 = TeDatumFactory:make("SAD69")
proj2 = TeLatLong:new(sad69)

-- Create a layer with a different projection
layer2 = TeLayer:new("Distritos_LL", db, proj2)

-- Copy the layer doing a remapping of its geometries
res = TeCopyLayerToLayer(layer1, layer2)
if (res) then
print("Layer Distritos was successfully copied to layer Distritos_LL")
else
print("Fail to remap layer!")
end
db:close()
return 0
```

Figura 4.5 – Código em Lua para copiar um layer.

```
require 'luaterralib'

-- Datatabase server parameters
host = "localhost"
dbname = "teste"
user = "root"
password = ""

-- Open a connection to the DB320RC1 MySQL database
db = TeMySQL:new()
if (db:connect(host, user, password, dbname) == false)
then
    print( "Error: ", db:errorMessage() )
end

print( "Connection successful to the database \"", dbname, "\" on MySQL server \"", host)

-- Load the layer "teste" which contains data of the districts of the Sao Paulo city
dist = TeLayer:new("teste")
if (db:loadLayer(dist) == false)
then
    print( "Fail to load the layer \"teste\": ", db:errorMessage() )
    db:close()


end

proj = dist:projection()

-- Create a view with the same projection of the layer
viewName = "SaoPaulo"

-- Check whether there is a view with this name in the datatabase
if (db:viewExist(viewName) == true)
then
    print( "There is already a view named \"SaoPaulo\" in the database\n")
    db:close()

end

view = TeView:new(viewName, user)
view:projection(proj)
if (db:insertView(view) == false)       -- save the view in the database
then
    print( "Fail to insert the view \"SaoPaulo\" into the database: ", db:errorMessage() )
    db:close()

end

-- Create a theme that will contain all of the objects of the layer (no restrictions applied)
theme = TeTheme:new("testeSaoPaulo", dist)
view:add(theme)

-- Set a default visual for the geometries of the objects of the layer
-- Polygons will be set with the blue color
color = TeColor:new()
polygonVisual = TeVisual:new(TePOLYGONS)
color:init(0,0,255)
polygonVisual:color(color)
theme:setVisualDefault(polygonVisual, TePOLYGONS)
-- Polygons will be set with the red color
pointVisual = TeVisual:new(TePOINTS)
color:init(255,0,0)
pointVisual:color(color)
pointVisual:style(TePtTypeX)
theme:setVisualDefault(pointVisual, TePOINTS)

-- Set all of the geometrical representations to be visible
allRep = dist:geomRep()
theme:visibleRep(allRep)
```

```
-- Set the attribute tables of the theme equal the tables of the layer
theme:setAttTables(dist:attrTables())

-- Save the theme in the database
    if (theme:save() == false)
    then
    print( "Fail to save the theme \"testeSaoPaulo\" in the database: ", db:errorMessage() )
    db:close()

end

-- Build the collection of objects associated to the theme
if (theme:buildCollection() == false)
then
    print( "Fail to build the theme \"testeSaoPaulo\": ", db:errorMessage() )
    db:close()


end
print( "The theme \"testeSaoPaulo\" was created without restrictions!\n")

-- Create a theme with the following attribute restriction:
-- Districts with population higher than 100,000 people

themeRest = TeTheme:new("Pop91GT100000", dist)
themeRest:setAttTables (dist:attrTables())

-- Set the attribute restriction
restAttr = " SPRNOME > 20 "
themeRest:attributeRest(restAttr)

-- Set all of the geometrical representations to be visible
themeRest:visibleRep(allRep)

-- Set the visual
themeRest:setVisualDefault(polygonVisual, TePOLYGONS)
themeRest:setVisualDefault(pointVisual, TePOINTS)

-- Insert the theme into the view
view:add(themeRest)

-- Save the theme in the database
if (themeRest:save() == false)
    then
    print( "Fail to save the theme \"Pop91GT100000\" in the database: ", db:errorMessage() )
    db:close()


end

-- Build the collection of objects associated to the theme
if (themeRest:buildCollection() == false)
then
    print( "Fail to build the theme \"Pop91GT100000\": ", db:errorMessage() )
    db:close()

end

print( "The theme  \"Pop91GT100000\" was created with attribute restrictions!\n\n")
db:close()
```

Figura 4.6 – Código em Lua para criar tema.

```
require 'luaterralib'

-- Datatabase server parameters
 host = "localhost"
 dbname = "teste"
 user = "root"
 password = ""

-- Open a connection to the teste MySQL database
db = TeMySQL:new()
if (db:connect(host, user, password, dbname) == false)
then
        print("Error: ",db:errorMessage())
        return
end
print("Connection successful to the database",dbname,"on MySQL server",host)
distritos = TeLayer:new("TesteLayer")
if (db:loadLayer(distritos) == false)
then
        print("Fail to load layer Distritos: ", db:errorMessage())
        db:close()
        return
end
-- Check whether the table of points to be generated
-- already exists in the database
 pointsTableName = distritos:tableName(TePOINTS)
if (pointsTableName == "")
then
        if (db:tableExist(pointsTableName))
        then
                print("The table of points",pointsTableName,"already exists in the database")
                db:close()
                return
        end
end

repPol = distritos:getRepresentation(TePOLYGONS)
centroids = TePointSet:new()  --generate centroids
if (db:centroid(repPol.tableName_, TePOLYGONS, centroids) == false)
then
        print("Fail to create centroids: ", db:errorMessage())
        db:close()
        return
end

--  Add new representation to the layer
distritos:addPoints(centroids)
print("Centroids created!")

db:close()
```

Figura 4.7 – Código em Lua para adicionar a representação de uma geometria.

5 Análise de resultados e trabalhos futuros

Este trabalho apresentou uma metodologia para aumentar a eficiência do processo de desenvolvimento de aplicações geográficas e para o ensino desse processo que envolve conhecimentos multidisciplinares. O uso de linguagens de alto nível e interpretadas para o desenvolvimento de aplicações geográficas, ao invés do uso direto de bibliotecas e frameworks em linguagens de programação de propósito geral, mostrou-se eficaz para o aumento da produtividade da equipe de desenvolvimento dessas aplicações. Os programadores puderam focar sua atenção e esforço em questões relativas ao domínio da aplicação ao invés de focar as complexas estruturas sintáticas e semânticas das linguagens de propósito geral.

Para comprovar os conceitos que suportam essa metodologia, a biblioteca C++ LuaTerraLib foi desenvolvida para estender a linguagem de script Lua com tipos, objetos e algoritmos providos pela biblioteca TerraLib para o desenvolvimento de aplicações SIG. O uso direto da biblioteca TerraLib requer um considerável conhecimento de C++ (templates, herança, polimorfismo), o que torna seu uso para o ensino das questões relativas ao processo de desenvolvimento de aplicações geográficas moroso e muito custoso. Para alunos de área não similares à Ciência da Computação como, por exemplo, Geografia e Engenharia Ambiental ou Geológica, seu ensino é quase inviabilizado. Como a curva de aprendizado da linguagem Lua é muito menor que a da linguagem C++, a biblioteca LuaTerraLib representa uma significativa contribuição para o ensino dos conceitos e tecnologias envolvidas no desenvolvimento de aplicações geográficas. O fato de Lua ser interpretada e apresentar baixo custo associado ao treinamento de novas equipes de desenvolvedores torna mais eficiente as fases de construção e teste do processo de desenvolvimento de software. Desta maneira, LuaTerraLib também se apresenta como uma contribuição para a rápida prototipagem de soluções SIG.

Como trabalhos futuros temos a finalização do projeto LuaTerraLib, visto que, apenas os módulos: Geometria vetorial, Raster, Elemento espaço-temporal, Metadado foram acabados. Os módulos: Funções, algoritmo geométrico e Armazenamento e recuperação do dado geográfico em SGBD-OR estão parcialmente acabados visto a complexidade de suas estruturas e compilação. Também devemos considerar a produção de uma metodologia para comprovar as melhoras no processo de desenvolvimento de software e no aprendizado da TerraLib.

6 Referências Bibliográficas

Câmara, G., Davis, C., & Monteiro, A. V. (2001). *Introdução a Ciência da Geoinformação.* Acesso em 15 de Setembro de 2008, disponível em Divisão de Processamento de Imagens - DPI: http://www.dpi.inpe.br

Câmara, G., Neves, M., Monteiro, A. M., Souza, R. C., Paiva, J. A., & Vinhas, L. (2002). SPRING and TerraLib: Integrating Spatial Analysis and GIS. *Proceedings of the SCISS Specialist Meeting*, (pp. 10-11). Santa Bárbara, CA, USA.

Câmara, G., Souza, R. C., Pedrosa, B. M., Vinhas, L., Monteiro, A. M., Paiva, J. A., et al. (2000). TerraLib; Technology in support of GIS innovation. *WORKSHOP BRASILEIRO DE GEOINFORMÁTICA, 2. (GEOINFO)* .

Crepani, E., & Medeiros, J. S. (2005). Imagens CBERS + Imagens SRTM + Mosaicos e o Cover LANDSAT em ambiente SPRING e TerraView: Sensoriamento Remoto e Geoprocessamento gratuitos aplicados ao desenvolvimento sustentável. *Anais XII Simpósio Brasileiro de Sensoriamento Remoto* .

Freeman, E., & Freeman, E. (2004). *Head First Design Patterns.* O'Reilly.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. R. (2004). *Padrões de Projeto – Soluções reutilizáveis de Software Orientado a Objetos.* Bookman.

Hirschi, A. (2007, Setember). Traveling Light, the Lua Way. *IEEE Software , 24*, pp. 31-38.

Ierusalimschy, R., Figueiredo, L. H., & Celes, W. (2007). The evolution of Lua. *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages* .

Manzur, A., & Celes, W. (2006). *tolua++ reference manual*. Acesso em 14 de Agosto de 2008, disponível em tolua++ reference manual: http://www.codenix.com/~tolua/tolua++.html.

Mascarenhas, F., & Ierusalimschy, R. (2008). Efficient compilation of Lua for the CLR. *Proceedings of the 2008 ACM Symposium on Applied Computing.*

Vinhas, L., & Ferreira, K. R. (2005). Descrição da TerraLib. In: M. Casanova, G. Câmara, L. Davis, L. Vinhas, & G. R. Queiroz, *Bancos de Dados Geográficos* (pp. 383-426). Curitiba: Editora MundoGEO.