14 Desenvolvimento de aplicativos com a TerraLib

Marcelo Tílio Monteiro de Carvalho
Mário de Sá Vera

14.1 Introdução

Este capítulo descreve o *TerraLib Development Kit - Tdk*, cujo objetivo principal é facilitar o desenvolvimento de aplicativos geográficos que utilizem a TerraLib.

A motivação do desenvolvimento do Tdk decorre de uma análise dos objetivos da TerraLib. Recorde do capítulo 12 que um dos principais objetivos do projeto TerraLib (www.terralib.org) é oferecer suporte, através de uma biblioteca (modelos, estrutura de dados e algoritmos), para a pesquisa e o desenvolvimento de tecnologias inovadoras na área da ciência da Geoinformação (ver Capítulo 12). Um segundo objetivo importante é estabelecer um ambiente colaborativo para o desenvolvimento de novas ferramentas e aplicativos que componham uma nova geração de Sistemas de Informações Geográfica (SIGs).

Para que este objetivo seja atingido de modo satisfatório, precisamos tratar três questões :

- A complexidade intrínseca do código da TerraLib demanda um tempo grande de aprendizado e requer experiência de programação (ver Capítulo 12).
- A TerraLib não oferecer suporte direto para o desenvolvimento de alguns componentes presentes em uma SIG básico. Em uma visão abrangente, pode-se indicar que este tipo de aplicação deve apresentar os seguintes componentes, apesar da sua implementação poder variar para cada sistema: interface gráfica-interativa com o usuário, entrada e integração de dados, armazenamento e recuperação de dados (organizados sob a

forma de um banco de dados geográficos), funções de consulta, funções de análise espacial e visualização e plotagem. Destes componentes, a TerraLib não oferece suporte direto para a interface gráfica-interativa com o usuário e para a visualização e plotagem.

- A possibilidade de interoperar dados com outras aplicações. Além de uma base de código aberto, uma comunidade de desenvolvimento de *software* deve poder utilizar padrões da indústria para troca de dados e serviços.

14.2 Motivação e estruturação de um SDK

14.2.1 Motivação para um SDK

Um SDK (*Software Development Kit*) é um conjunto de ferramentas que permite o desenvolvimento de aplicativos utilizando-se funcionalidades de um *software* ou de uma biblioteca referente a um domínio de aplicação específico. Um SDK é justificado quando queremos oferecer as funcionalidades de uma ferramenta, criada por especialistas em um domínio específico, para acelerar o desenvolvimento de aplicativos que dependam de funcionalidades deste domínio. Normalmente, um SDK oferece APIs (*Application Programming Interfaces*) ou *plugin*, além de utilitários que agilizam a programação, como ferramentas para ajudar na depuração e, documentação detalhada. Um exemplo de um SDK para SIGs são as extensões do ArcGIS da ESRI (www.esri.com/software/arcgis).

Uma API é um conjunto de definições que permitem acesso às funcionalidades e serviços de um determinado *software* ou biblioteca. Este conjunto pode ser disponibilizado como funções de uso comum (ex. funções para desenhar janelas e ícones), permitindo que programadores não precisem programar tudo do início. Uma boa API oferece um alto nível de abstração escondendo detalhes da implementação.

Um *plugin* é um pedaço de programa desenvolvido para oferecer uma funcionalidade específica. É um *mini-programa*, que roda embutido e utilizando a interface de um programa principal. Normalmente, os *plugins* possuem uma definição dos limites de suas funcionalidades Um exemplo de *plugin* é o SVG da Adobe (www.adobe.com/svg).

De uma forma resumida, as principais vantagens de um SDK são:

- Permitir que os aplicativos possam ser desenvolvidos em menos tempo e em menos passos.
- Oferecer facilidades para construção de aplicações sem a necessidade de muita experiência em programação.
- Oferecer suporte a várias linguagens de programação.
- Facilitar a formação e manutenção de uma comunidade. Através de um SDK, a comunidade pode contribuir com sua criatividade.

14.2.2 Estruturação de um SDK

Para poder oferecer as vantagens citadas acima, um SDK deve ser estruturado respeitando alguns requisitos:

- Modularidade – o acoplamento entre os componentes oferecidos por um SDK deve ser pequeno, para que o programador tenha liberdade para utilizar somente os componentes que ele necessite.
- Reuso – os componentes de um SDK devem ser extensíveis, para que o programador possa, com alguma adaptação, reutilizá-los, de acordo com as necessidades específicas da sua aplicação. Além disto, devem poder ser utilizados em diversos contextos e realizar diferentes tarefas.
- Interface com múltiplas linguagens de programação – é desejável que um SDK disponibilize APIs para mais de uma linguagem de programação. Para isto, O SDK deve ser pensado como uma especificação genérica, a partir da qual as APIs devem ser implementadas. Isto garante a compatibilidade entre as implementações, ao mesmo tempo que mantém a independência.

14.2.3 O TerraLib Development Kit

Conforme o exposto acima, fica evidente que seria desejável a TerraLib possuir um SDK. Deste modo, iniciamos o projeto Tdk (*TerraLib Development Kit*), cujo objetivo principal é facilitar o desenvolvimento de aplicativos geográficos que utilizem a TerraLib.

Para atingir este objetivo, projetamos o Tdk levando em consideração os seguintes requerimentos:

- API Simplificada – prover, para usuários que não sejam proficientes em programação com a TerraLib, uma API simplificada para acesso às suas funcionalidades mais comuns. O Tdk, no entanto, não oferece a mesma flexibilidade que o código da TerraLib, de forma que deve ser permitido o acesso direto à TerraLib.
- Arquitetura genérica – a arquitetura do Tdk não deve depender de nenhuma tecnologia específica (aberta ou proprietária). Idealmente o programador deve poder implementar a arquitetura proposta com a tecnologia mais apropriada ao contexto da sua aplicação (Web ou Desktop, JAVA ou C++). De uma forma simplista, a arquitetura do Tdk consiste em uma especificação de como implementar um ambiente de desenvolvimento para SIGs.
- Modelos reutilizáveis e extensíveis – sugerir modelos funcionais que sejam úteis para o desenvolvimento de SIGs (modelos para autenticação de usuários, acesso ao banco de dados através de múltiplas conexões, modelos para edição e impressão de mapas, entre outros). As soluções propostas pelos modelos sugeridos devem poder ser usadas isoladamente e poder ser estendidas, para acomodar funcionalidades específicas da aplicação dos usuários.
- Compatibilidade com padrões – oferecer uma interface para o acesso à TerraLib, compatível com os padrões publicados pelo Open GIS Consortium (OGC) (www.opengis.org). Isto permite que os programadores acostumados com o vocabulário e a arquitetura do OGC, utilizem as funcionalidades do Terralib no desenvolvimento de suas aplicações. Além disto, para promover a interoperabilidade com aplicativos desenvolvidos com a TerraLib, o Tdk deve oferecer serviços compatíveis com as especificações do OGC (como os serviços WMS, WFS e WCS, de publicação de dados na Web).

Com isto, esperamos resolver as três questões levantadas acima e facilitar o desenvolvimento de um ambiente colaborativo para o desenvolvimento de SIGs avançados.

14.3 Fundamentos conceituais da arquitetura

Desenvolvemos o Tdk como uma extensão modular da TerraLib, direcionado a oferecer suporte para a implementação de funcionalidades de aplicativos no domínio de SIG. Isto se reflete no modelo de dados utilizado, concebido a partir do modelo de dados original da TerraLib. Embora seja desenvolvido em cima da biblioteca TerraLib, o Tdk não restrinje o acesso direto às suas funcionalidades, conforme mostrado na Figura 14.1.

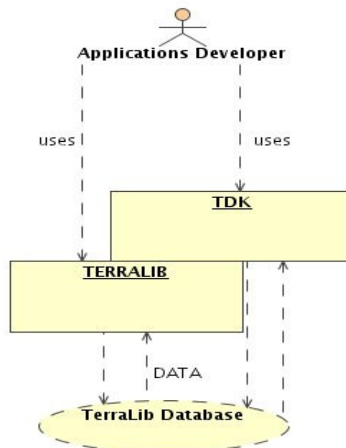

Figura 14.1 – O Tdk como extensão modular da TerraLib.

A arquitetura do Tdk especifica interfaces de programação (APIs) bem definidas e sugere modelos funcionais referentes à implementação de funcionalidades comuns a aplicativos gráfico-interativos de SIGs. Os elementos da arquitetura são descritos a seguir.

14.3.1 Modelo de dados Tdk

Definimos o modelo de dados conceitual do Tdk a partir do modelo original da TerraLib (tema, layer, view, etc. Ver Capítulo 12). O modelo do Tdk segue as especificações de um *composite* (Gamma et al., 1995) e pode, portanto, ser estendido para acomodar novos componentes, conforme mostrado na Figura 14.2.

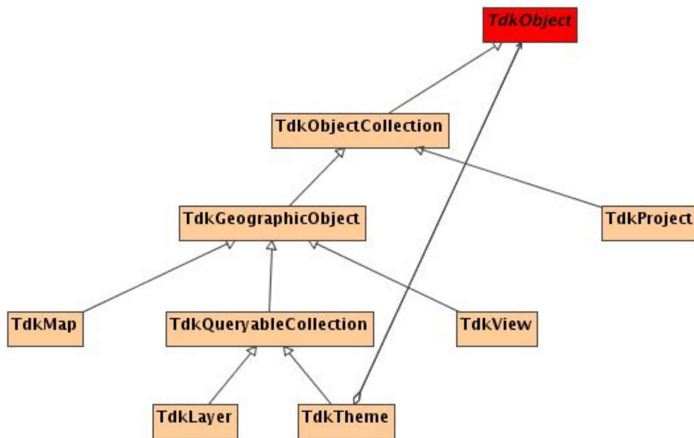

Figura 14.2 – Diagrama hierárquico com o modelo de dados Tdk.

O elemento que define um objeto Tdk (*TdkObject*) é a raiz do modelo e pode representar três tipos básicos de elementos:

- Geometria – Uma representação geométrica simples (um ponto, linha, polígono, imagem, ou qualquer outra representação geométrica).
- Objeto Simples – Um objeto geográfico composto por n geometrias em atributos.
- Coleção – Um conjunto de objetos simples ou de outras coleções.

Esses elementos são organizados hierarquicamente segundo as regras de que um objeto simples agrupa um conjunto de geometrias e uma coleção agrupa objetos simples ou outras coleções. Uma vez definida uma interface única e comum a objetos e coleções, torna-se possível o tratamento uniforme entre estes elementos.

14.3.2 Interfaces de programação

Como concebemos o Tdk dentro do paradigma de orientação a objetos (Rumbaugh et al., 1991), a presença de componentes na arquitetura é naturalmente esperada. Neste contexto, componentes são classes projetadas a partir do conceito de reuso e que são acessados através de

métodos e operações especificadas pela interface do componente. Os componentes devem ser extensíveis. O Tdk oferece um conjunto de componentes agrupados em uma interface de programação.

Notamos, no entanto, que em muitos casos, o uso devido de componentes requer um conhecimento grande sobre como utilizá-los, o que pode levar o programador a um aprofundamento em documentos técnicos. Como alternativa, visando diminuir o tempo necessário para se obter proficiência na utilização do Tdk, projetamos uma segunda interface de programação, com características de programação procedural, através de métodos que se comportam como funções. Esta interface oferece um conjunto de funcionalidades bem definidas, com um alto nível de abstração, escondendo a complexidade da implementação. Os métodos oferecidos, chamados de serviços, foram agrupados segundo seu contexto semântico, de forma que as funcionalidades possam ser facilmente localizadas e acessadas diretamente. Diferente dos componentes, os serviços não podem ser estendidos.

Em resumo, o Tdk oferece duas interfaces de programação (ver Figura 14.3), conforme descrevemos a seguir :

- uma API de componentes, através da qual se pode acessar um conjunto de classes que encapsulam dados e operações definindo comportamento consistente a ser utilizado ou estendido;
- uma API de Serviços, na qual as funcionalidades mais comuns são agrupadas de acordo com o seu contexto e a sua semântica e são acessíveis através de uma interface simplificada. Idealmente, a API de serviços deve referenciar tipos primitivos nas assinaturas de seus métodos, de modo a não ser necessário conhecimento prévio sobre componentes do Tdk, embora internamente os serviços devam utilizá-los.

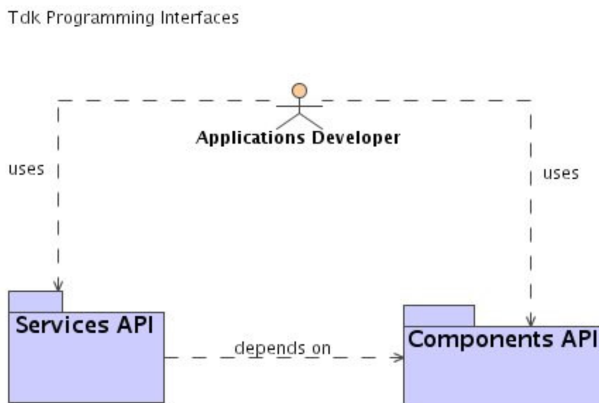

Figura 14.3 – As interfaces de programação Tdk.

14.3.3 Modelos funcionais

Os modelos funcionais sugerem estratégias de implementação para as funcionalidades de um sistema. No Tdk, projetamos modelos para atender às principais funcionalidades presentes em um tipo de aplicativo comum dentro do domínio de um SIG. Neste sentido, estamos falando de aplicações gráfico-interativas, que persistem seus dados em um banco de dados e que oferecem operações básicas de visualização, impressão, consulta e edição.

Para atender a este tipo de aplicação, o Tdk oferece três modelos principais: i) um modelo de persistência, que gerencia a comunicação com o banco de dados, ii) um modelo de interação, que estrutura a comunicação entre os elementos do sistema e possibilita a implementação das funcionalidades de visualização, impressão, consulta e edição de uma forma integrada e iii) um modelo de apresentação, que define um conjunto de componentes GUI, implementadas sobre um pacote gráfico-interativo externo. Existe ainda a intenção de oferecer um modelo temporal, para suporte à aplicações dinâmicas.

A concepção dos modelos funcionais no Tdk visa atender a um requisito de extensibilidade e reuso, oferecendo a possibilidade do programador adaptá-lo às suas necessidades específicas.

Em termos de implementação, os modelos funcionais serão construídos utilizando-se as interfaces de programação Tdk, conforme ilustrado na Figura 14.4.

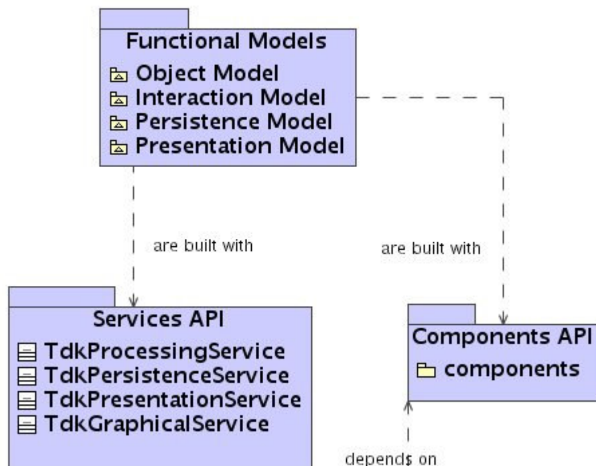

Figura 14.4 – Os Modelos funcionais são implementados utilizando-se as interfaces de programação Tdk.

14.4 Descrição técnica dos elementos da arquitetura

Nesta seção estaremos refinando as definições apresentadas como fundamentos da arquitetura Tdk. Descrevemos, em detalhes, os principais elementos referenciados até aqui. Com objetivo didático, estaremos exemplificando, através de codificação, o uso de alguns destes elementos.

14.4.1 API de componentes

Alguns dos principais componentes oferecidos pelo Tdk são descritos abaixo:

Componente referente a um Objeto Tdk (TdkObject) – este componente consiste na raiz da árvore hierárquica de dados Tdk. Ele representa a

agregação das interfaces que impõe comportamento a todos os elementos do modelo de dados Tdk (ex: TdkEventHandler, TdkPersistenceObject).

Componente para interface de persistência (TdkPersistenceObject) – define o comportamento de uma classe aderente ao modelo de persistência Tdk. As classes de aplicação deverão implementar esta interface direta ou indiretamente. As classes pertencentes ao modelo dados Tdk implementam indiretamente esta interface através do TdkObject.

Componente de identificação de objetos (TdkGlobalID) – a idéia de tratar todos os elementos, coleção ou folha, de uma base de dados geográfica da mesma forma abre espaço para a criação de um identificador único para cada TdkObject. O Global ID (GID) foi criado com esse propósito. A idéia do GID é representar TkdObjects da mesma forma que o sistema de DNS representa URL's, dessa forma podemos ter um GID único para cada TdkObject do sistema (inclusive sistemas que englobam vários bancos de dados). O GID é composto de: [Database Descriptor].[Object Type].[Object ID]

Componente para fabricação de objetos (TdkObjectFactory) – responsável por criar instâncias de TdkObjects dinamicamente. Todos os tipos criados por uma aplicação, que possuírem uma classe representante definida pela própria aplicação, devem ter métodos construtores registrados nesta classe. Ela contém uma estrutura de dados que tem o nome da classe como chave e um ponteiro para uma função que cria uma instância da classe correspondente ao tipo.

Componente Canvas Genérico (TdkCanvas) – representa uma abstração, independente de pacotes gráficos, de um canvas (painel) para desenho que tem formas primitivas para a TerraLib. Em outras palavras, o TdkCanvas provê uma interface específica para o desenho de geometrias TerraLib, que deve ser estendida para ter acesso às suas funcionalidades (ex. plotPoint, plotLine, plotPolygon, plotText, plotRaster).

Como exemplo de implementação de um *canvas* genérico mostramos aqui o *TdkCDCanvas*. Este componente consiste na versão CD (www.tecgraf.puc-rio.br/cd) de um *TdkCanvas*. No código apresentado a seguir mostramos como desenhar um ponto simplesmente :

```
/* declarações omitidas :
   •  qtParent -  é a janela Qt principal, na qual,este canvas
      estará associado.
   •  object – um objeto Tdk com geometria representada por um
      conjunto de polígonos.
   */
//…
TdkQtCanvas* canvas = new TdkQtCanvas(qtParent);
//Armazena a geometria num container.
TePolygonSet& polys=
((TdkGeographicObject*)object)->getPolygonsGeometry();

for(int j = 0; j < polys.size(); j++)
   canvas->plotPolygon(polys[j]);
//…
```

Codificação 14.1 – Exemplo de implementação em C++ do componente *TdkCanvas* com o pacote gráfico CD.

Componente de acesso ao banco de dados (TdkDatabase) – os dados e metadados são armazenados no banco TerraLib em tabelas relacionais e refletidos nas classes definidas. As tabelas são divididas em dois grupos; tabelas de metadados, que são usadas para guardar o conceito Terralib e possuem formato pré-definido e as tabelas de dados, que são usadas para armazenar os dados geográficos (componente espacial + descritiva). A TerraLib implementa um primeiro nível de abstração destes dados com a interface `TeDatabase` (ver Capítulo 12), onde temos métodos para manipular geometrias, layers, temas, etc.. Porém, é a interface *TdkDatabase* que oferece um nível mais alto de abstração, introduzindo tipos como *TdkMap* e *TdkProject*, e escondendo do programador detalhes do modelo de dados TerraLib.

Como exemplo de implementação do componente *TdkDatabase* apresentamos, a seguir, um código de acesso a uma tabela existente em um banco de dados Oracle :

```
//...
TdkOracleConDescriptor desc("oraserver",
                            "tdkuser",
                            "tdk123" ,
                            "gisdev");
TdkDatabase* oracleDriver = new TdkOracleImpl(desc);
String tableName = "myTable";
oracleDriver->loadTable(tableName);
//…
```

Codificação 14.2 – Exemplo de implementação em C++ de *TdkDatabase* para acesso a um banco Oracle.

Abstração de uma aplicação básica (TdkAplication) – provê métodos que oferecem funcionalidades básicas para agilizar o processo de desenvolvimento de um SIG básico, como a criação de um novo modelo de dados, de um novo projeto, funções para visualização e consulta, impressão, etc.. O TdkApplication não provê métodos de interface gráfica, de modo que o programador pode escolher o *toolkit* para interface de sua preferência.

Como exemplo de implementação do componente *TdkApplication* apresentamos o código a seguir, que configura o modo de interação de uma aplicação que utiliza o pacote gráfico IUP (www.tecgraf.puc-rio.br/iup) para responder à interação com o usuário de forma a criar geometrias – linhas no caso – em uma base de dados TerraLib :

```
//…
Ihandle* btn = IupButton("", "CreateLineCb");
IupSetAttributes(btn,
                 "TIP = \"Criar linha\",
                 IMAGE = lines_image,
                 IMPRESS = lines_press_image,
                 IMINACTIVE = lines_inactive_image,
```

```
                    PRESSED = 0");
IupSetFunction("CreateLineCb",
                (Icallback)TdkIupApplication::CreateLine);
//…
```

Codificação 14.3 – Exemplo de implementação em C++ do componente *TdkApplication* sob o pacote gráfico IUP.

Componente de controle de interação (TdkController) – componente raiz da hierarquia de componentes responsáveis pela implementação da estratégia de resposta às ações do usuário. Todas as controladoras descritas adiante, no modelo de interação, irão implementar esta interface (*TdkMapController,TdkInteractController etc...)*.

14.4.2 API de serviços

Alguns dos principais serviços oferecidos pelo Tdk são descritos abaixo:

Serviço de persistência – disponibiliza funcionalidades de alto nível que permitem persistir, consultar e atualizar as informações em um banco de dados TerraLib.

O código a seguir ilustra o que é necessário para criar uma base de dados TerraLib completa, utilizando o Tdk. Modificar o código para receber outro banco de dados, como o Oracle por exemplo, seria um trabalho muito simples, bastando trocar o descritor de conexão para o descritor Oracle (*TdkOracleConDescriptor*) de acordo com os parâmetros necessários para esta conexão (nome do servidor etc...).

```
//…
TdkAccessConDesc desc(C:/“mydb.mdb”);
TdkPersistenceService.createTableModel(desc);
//…
```

Codificação 14.4 – Exemplo de uso do Serviço de persistência em Java na criação de um banco de dados TerraLib.

Serviço de processamento – provê funcionalidades que auxiliam as tarefas de calcular, converter dados e selecionar áreas georeferenciadas.

Serviço gráfico – oferece uma série de funcionalidades para interface gráfica baseadas no *TdkCanvas*.

Serviço de apresentação – oferece métodos para criação de diálogos de comunicação com o usuário.

14.4.3 Modelo de persistência

Conforme mencionado na Seção 1.2.3, o Tdk oferece um modelo funcional para a implementação do processo de gerência da persistência de dados em uma base TerraLib. Este modelo explora, principalmente, os conceitos de *transparência de localidade* e *automação da persistência*.

- Transparência de localidade – este conceito consiste em poder tratar dados de maneira uniforme, independente da sua origem.
- Automação da persistência – este conceito consiste em assumir a responsabilidade de controlar o acesso e atualização dos dados a serem persistidos.

Para implementar o conceito de transparência de localidade, o modelo define o uso de um identificador global (*TdkGlobalID*) por todas as classes aderentes ao modelo de persistência. Nesta identificação, a classe carrega a informação referente à sua base TerraLib de origem como descrito na especificação funcional do Tdk (www.terralib.org/tdk). No domínio de SIGs, podemos imaginar uma aplicação deste conceito quando se deseja visualizar dados oriundos de bancos de dados distintos.

Para implementar o conceito de automação da persistência, foi criado um componente *TdkPersistenceObject,* que define uma interface única com os métodos a serem implementados por uma classe nova interessada em aderir ao modelo de persistência. Este componente é representado por um novo elemento, que foi acrescentado ao modelo de dados original (ver Figura 14.4), conforme mostrado na Figura 14.5.

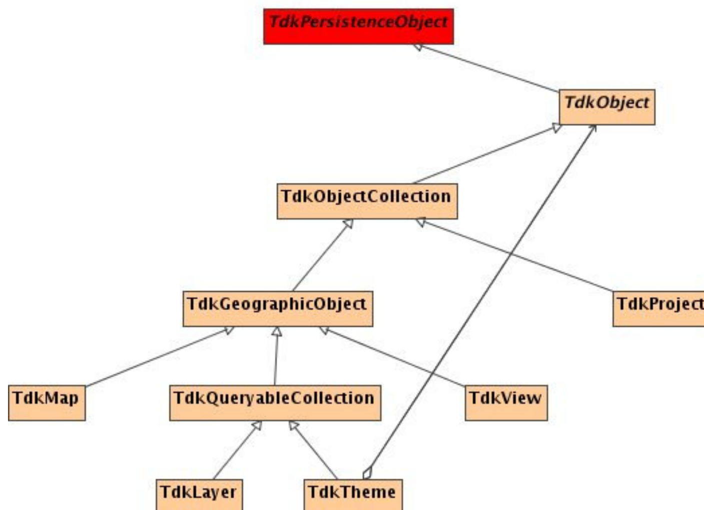

Figura 14.5 – O modelo de dados Tdk incluindo a classe *TdkPersistenceObject*.

Através da implementação deste último conceito, a tarefa de persistência dos dados da aplicação é muito simplificada. Duas situações podem ocorrer: i) o conjunto de elementos que definem o modelo conceitual Tdk é suficiente para atender às demandas do aplicativo a ser desenvolvido. Neste caso, o programador não precisa se preocupar em implementar nenhum código relacionado à persistência de seus dados numa base TerraLib. ii) o modelo original do Tdk é estendido através de novos elementos necessários ao domínio da aplicação. Neste caso, os componentes que representam os novos elementos devem implementar a interface única definida pelo *TdkPersistenceObject*. Desta forma, estes componentes passam a aderir ao modelo de persistência, ganhando a possibilidade de explicitar as particularidades referentes à persistência de seus atributos na base TerraLib. Sem a utilização do conceito de automação, seria necessário estender o componente da TerraLib

(*TeDatabase*) e definir os métodos de persistência para os novos elementos[1], que representa uma tarefa bem mais complicada.

Uma outra questão tratada pelo modelo de persistência, se refere ao controle do ciclo de vida de componentes provenientes da extensão do modelo de dados do Tdk. Como fazer, por exemplo, para que ao carregar um tema TerraLib, composto por objetos definidos fora do modelo de dados Tdk, seja possível instanciar esses objetos de forma automática?

Para endereçar a situação descrita acima, o Tdk implementa o conceito de uma fábrica de objetos (ver Seção 1.3.1), que possibilita às classes aderentes ao modelo de persistência registrar seus atributos simples e tê-los, desta forma, gerenciados pelo Tdk.

Apresentamos a seguir um exemplo da utilização do modelo. Na codificação ilustrada abaixo realizamos a carga de um tema persistido para a manipulação em memória :

```
TdkAccessConDescriptor desc( "C:/mydb.mdb" );
TdkObjectGID tdkObjGID(1,
                        "1",
                       "TDK_ THEME",
                       desc.getDbKey());
TdkTheme myTheme( tdkObjGID );
TdkPersistenceService::loadObject( &mTema );
```

Codificação 14.5 – Carregando um objeto persistido para memória.

14.4.4 Modelo de interação

Conforme mencionado na Seção 1.2.3, pretendemos atender à demandas que surgem no desenvolvimento de aplicativos gráfico-interativos. Para isto, projetamos um modelo, que sugere estratégias de implementação para o suporte às operações que requeiram interação com o usuário. O modelo de interação do Tdk apresenta as seguintes características:

---

[1] Como foi feito com *TdkProject*.

- Flexibilidade - permite uma implementação genérica e independente de soluções proprietárias. É independente também do pacote gráfico a ser utilizado;
- Independência modular – através da utilização de um modelo de eventos, permite um desacoplamento entre os seus componentes;
- Para garantir estas características, concebemos o modelo com base no padrão MVC de arquitetura (ver Figura 14.6) (Krasner e Pope, 1988).

MVC (Model,View,Controller) – este padrão é bastante difundindo no domínio de aplicações gráfico-interativas e tem como principal objetivo organizar a interação entre os elementos participantes da implementação de uma determinada funcionalidade, garantindo o desacoplamento entre a interação do usuário e a representação do dado.

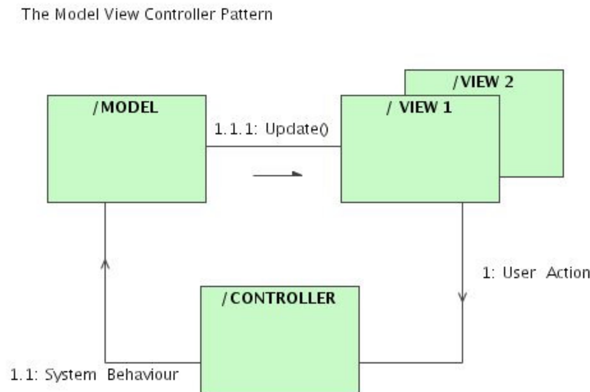

Figura 14.6 – O padrão MVC para desenvolvimento de aplicativos de visualização.

Uma vez aplicado, o padrão *MVC* resulta em uma divisão em camadas de contextos – Modelo, Visão e Controle – onde cada camada possui um papel bem definido na interação. No Modelo estarão os dados do domínio representado (ex: temas e objetos). A Visão representa a interface direta com o usuário – como uma metáfora de um conceito do

domínio (ex: legenda, mapa). Finalmente, o Controle determina a reação à uma ação do usuário sobre os dados do Modelo (ex: *zoom, pan*).

Como exemplo de aplicação do *MVC,* podemos imaginar, a visualização de diversos mapas em uma aplicação de visualização de dados geográficos e, no entanto, todas essas visualizações fazerem referência ao mesmo dado representado na camada de dados que, por sua vez, representa o dado persistido no banco de dados.

A Figura 14.7 apresenta a divisão das camadas de classes adotada pelo modelo de interação do Tdk, que são descritas em seguida.

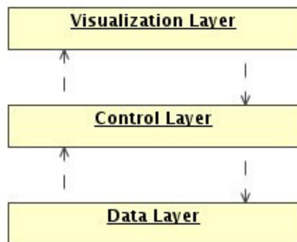

Figura 14.7 – As camadas referentes ao MVC do modelo de interação

Camada de Visualização (Visualization Layer) – nesta camada apresentamos um conjunto de metáforas do domínio da aplicação, responsáveis pela interação direta com o usuário. Não nos referimos aqui ao controle desta interação, mas sim a contextualização das possíveis ações que faz sentido realizar em cada entidade representada nesta camada (ex: uma legenda pode ter a ordem de seus temas trocada). Seguindo o padrão *MVC,* nesta camada, estariam as visões (*View*) dos modelos (*Model*) pertencentes à camada de dados.

Como exemplos de componentes desta camada podemos citar o componente de visualização de um tema TerraLib responsável por encapsular atributos de visualização como estilos de objetos selecionados.

Camada de Controle (Control Layer) – nesta camada acontece a mediação da interação entre usuário e aplicação. As ações do usuário, realizadas via

elementos da camada de visualização, realizam algum tipo de operação sobre os dados representados, logicamente, pela camada de dados. A estratégia de reação, por parte do sistema, às ações do usuário são implementadas nesta camada. Estas estratégias são, inclusive, configuráveis de acordo com o tipo de comportamento desejado na resposta do sistema. Exemplificando, um *mouse click* pode causar uma reação de criação de ponto ou seleção de objeto, dependendo da configuração da camada de controle. Seguindo o padrão *MVC,* nesta camada estão as controladoras (*Controllers*) que definem o comportamento do sistema em reposta às acoes do usuario sob as visões (*Views*) dos dados (*Model*).

Como exemplos de componentes desta camada podemos citar dentre outros :

- A controladora de interação com o Mapa visualizado
- A controladora de interação com a Legenda

Uma observação final quanto a camada de controle do modelo de interação é a adoção de um paradigma de programação orientada a eventos. Apesar do *MVC* não depender deste paradigma, já que seria possível implementá-lo utilizando chamadas de métodos simples (*callback methods*), esta estratégia nos trouxe uma flexibilidade interessante. Com o uso de eventos, o código responsável pela notificação de uma determinada ocorrência (ex: a troca de estilo de um objeto), assim como o código que implementa a resposta à uma ação do usuário (ex: *mouse move*), ficam completamente separados do código funcional (*business code*). O modelo de eventos adotado foi sugerido pela arquitetura VIX (Santos, 2005) (Ver Figura 14.8).

A arquitetura VIX – A maioria dos sistemas gráfico-interativos adota o modelo de orientação a eventos, difundido a partir de Smalltalk (Goldberg e Robson, 1983). Tipicamente, o fluxo de informações neste tipo de aplicação segue um padrão. A cada ação do usuário, são gerados eventos do sistema de interface que passam o controle para a aplicação.

Com o intuito de facilitar o desenvolvimento de aplicativos com este tipo de interação, é desejável tratar os eventos de maneira uniforme, independente do tipo de ação que ele irá ocasionar. Isto levou à adoção do conceito de objetos visuais interativos (VOs).

Pode-se entender um objeto visual como qualquer objeto que possua uma representação e um comportamento.

Outro conceito importante é o de espaço visual (VS), que mapeia entidades que transmitem eventos para VOs e fornecem uma superfície de visualização onde eles são posicionados. Os VSs devem ser encarados como elementos de ligação entre o usuário e o VO que ele está manipulando. No processo de comunicação entre estes objetos, são utilizados dois mecanismos:

- Mensagens - através deste mecanismo os objetos podem trocar dados uns com outros (o VO deve saber responder às requisições do VS).
- Filtros - servem para modelar objetos que fazem a interface entre um VS e um VO. Um filtro repassa para o seu VO associado os eventos gerados em seu VS, podendo dar algum tratamento especial a esses eventos.

Na implementação Tdk da arquitetura VIX, como parte estrutural do modelo de interação definimos dois novos elementos:

- uma interface única para encapsular o código referente a troca de mensagens entre os componentes (TdkEventHandler).
- uma abstração de um objeto de edição (TdkLayoutObject) que servirá de componente raiz para todos os elementos do modelo de dados Tdk presentes na implementação de funcionalidades de edição.

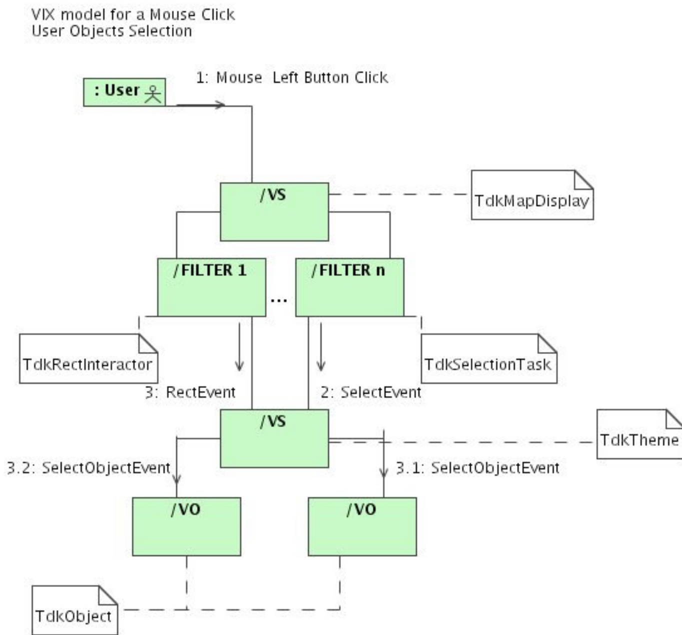

Figura 14.8 – O modelo de eventos baseado no VIX

Apresentamos a seguir o modelo de dados do Tdk, com estes novos elementos.

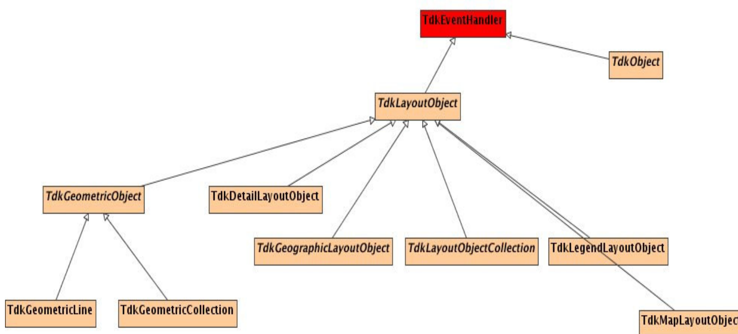

Figura 14.9 – O modelo de dados Tdk acrescido dos elementos referentes ao modelo de interação.

Para efeitos de ilustração, mostramos abaixo uma implementação de resposta ao evento de *scrolling* (rolagem da barra de janela da interface com o usuário) do usuário, tratado pela controladora do mapa visualizado .

```
void TdkMapController::
handleVSEvent(TdkScrollEvent& event)
{
   float x = event.getX();
   float y = event.getY();
   double dx = event.getDX();
   double dy = event.getDY();
   double xc = x + dx / 2.0;
   double yc = -y - dy / 2.0;
   mapDisplay_->center(xc, yc);
   mapDisplay_->draw();
}
```

Codificação 14.6 – Tratamento de um *scroll* no visualizador de mapas pela classe controladora *TdkMapController*.

Camada de Dados (Data Layer) – o principal objetivo da camada de dados é o armazenamento lógico dos dados persistidos.Em termos de interação com as outras camadas, ela é manipulada pela camada de visualização e controle através do modelo de eventos.Seguindo o padrão *MVC* esta camada armazena os componentes de modelo (*Model*).

Como exemplo de um componente desta camada podemos citar a representação de um tema, armazenando em memória seus objetos e atributos.

14.4.5 Modelo de apresentação

Este modelo apresenta um conjunto de componentes GUI, implementadas sobre um pacote gráfico-interativo externo. Os componentes GUI são implementados da forma mais fina possível, tendo comportamento e estado independentes de recursos específicos do pacote

gráfico. Desta forma, pretende-se minimizar o esforço diante de uma mudança de pacote gráfico. Além disto, tendo estado e comportamento descritos na camada funcional, a aplicação passa a ter mais flexibilidade sobre o comportamento do modelo. Como exemplos de componentes desta camada podemos citar:

- Diálogo para apresentação de atributos de um objeto.
- Menu de interação com o usuário.
- Diálogo para importação de dados.

14.5 Compatibilização com o OGC

Conforme mencionado na introdução (Seção 1.1.3), um dos requerimentos do Tdk é oferecer suporte a padrões do OGC. Este suporte é oferecido de duas formas:

- Uma camada de interface para o modelo de dados do OGC, que permite os programadores acostumados com o vocabulário e a arquitetura do OGC utilizarem as funcionalidades do Terralib.
- Serviços do OGC. Um exemplo disto é o serviço WMS para publicação de dados na web.

14.5.1 Interface para programação com o OGC

O Tdk oferece uma API que mapeia o modelo de dados do Tdk para o modelo de dados do OGC (www.opengeospatial.org). Sendo assim, é possível o desenvolvimento de aplicativos que utilizem as funcionalidades oferecidas pela TerraLib e que sejam compatíveis com os padrões publicados pelo OGC (Figura 14.10).

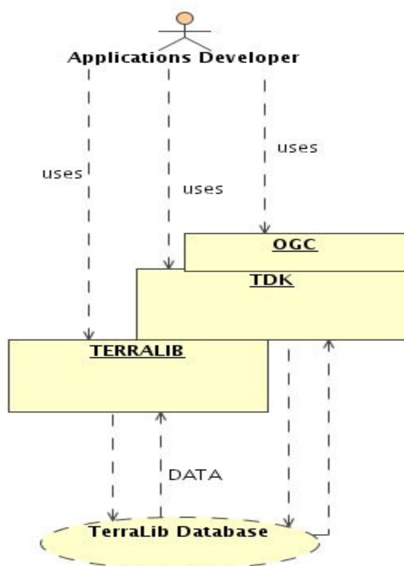

Figura 14.10 – A interface do Tdk com o padrão OGC.

14.5.2 Serviço WMS

O serviço WMS (*Web Map Servise*) permite a publicação de mapas na Web, produzidos a partir de dados georeferenciados. O serviço especifica um padrão de como o cliente deve requisitar as informações para o servidor e como este deve responder. Para esta comunicação são definidas três operações: *GetCapabilities*, *GetMap* e *GetFeatureInfo* (www.opengeospatial.org).

- *GetCapabilities*: permite ao cliente solicitar todas as metainformações sobre a base de dados, tais como o nome dos mapas contidos, projeção, escala, estilo possíveis dos mapas, formatos possíveis dos mapas (GIF, JPEG, PNG), formato das informações descritivas (FeatureInfo). Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo XML (text/xml) no mesmo protocolo. Segundo o protocolo WMS, esta operação deve ser implementada obrigatoriamente.
- *GetMap*: retorna o mapa propriamente dito. Com esta operação, o cliente do Web Service pode requisitar um mapa em particular. Para

isto basta usar as informações que foram retornadas pelo GetCapabilities. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo de imagem no formato solicitado, que pode ser um das opções: GIF, PNG ou JPEG. Esta operação também é obrigatória.

- *GetFeatureInfo*: permite ao cliente realizar consultas nos mapas da base de dados. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo no formato solicitado que pode ser um dos seguintes: texto, XML ou GML. Esta operação não é obrigatória segundo o protocolo.

14.6 Exemplos

A especificação definida pelo Tdk pode ser implementada em diversos contextos. Mostramos abaixo alguns exemplos, onde ilustramos o uso dos elementos de arquitetura Tdk na confecção de diferentes aplicativos.

14.6.1 Um sistema para visualização, impressão, consulta e edição

Ilustramos a seguir aspectos do processo de desenvolvimento de um aplicativo gráfico-interativo, que possui operações de visualização, impressão, consulta e edição de dados geográficos, a partir do Tdk. O processo consiste em estender as classes definidas pelo modelo de interação (ver Figura 14.11) para atender às funcionalidades da aplicação (zoom, impressão, criação de pontos e linhas, seleção de objetos, etc.) e a utilização dos modelos de Persistência e Apresentação.

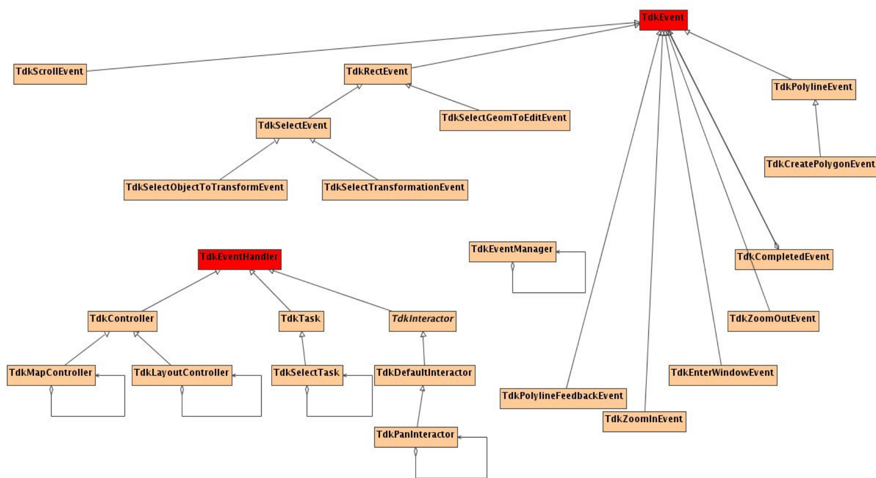

Figura 14.11 – A extensão do modelo de interação para o sistema VICE, onde capacitamos a aplicação com funcionalidades de seleção*, zoom*, criação de geometrias.

O aplicativo VICE - Sistema para Visualização, Impressão, Consulta e Edição – oferece a noção de modos de operações de acordo com o contexto de utilização do aplicativo (Visualização e Consulta, *Layout* e *Preview* de Impressão. A utilização do modelo funcional de interação, foi fundamental para a implementação destes modos de operações. A seguir, apresentamos as telas (*printscreen*) referentes a cada um desses modos em funcionamento (Figuras 14.12, 14.13 e 14.14).

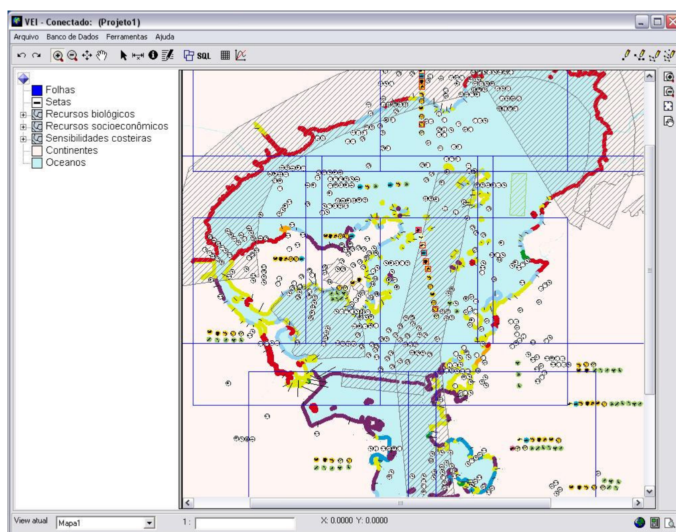

Figura 14.12 – VICE - modo de Visualização e Consulta.

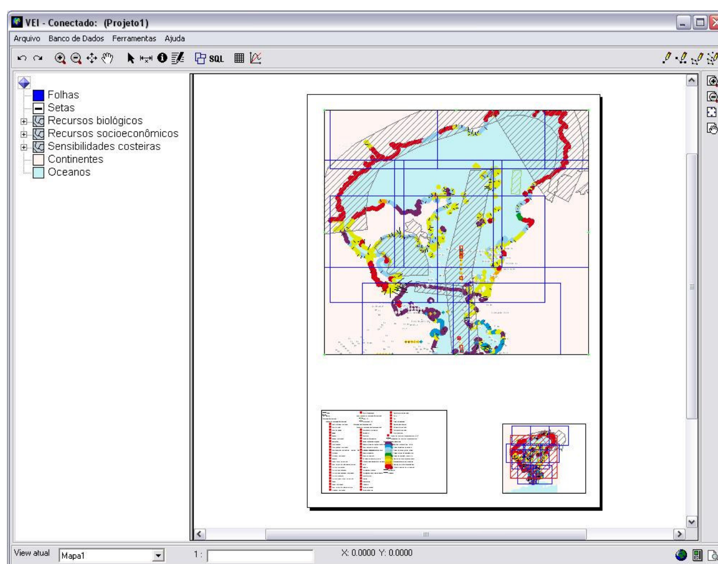

Figura 14.13 – VICE - modo de *Layout*.

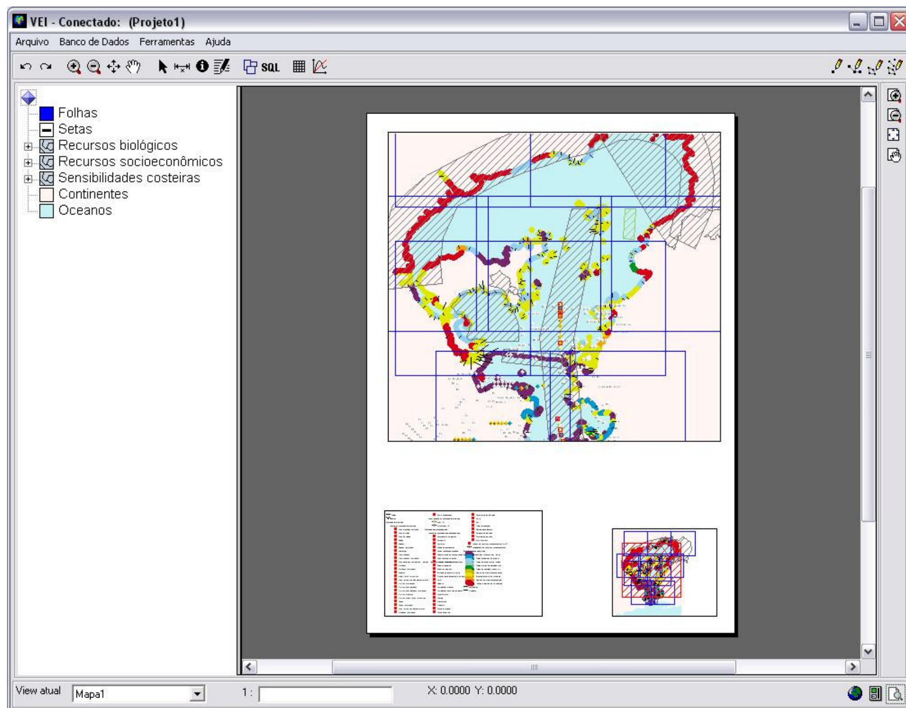

Figura 14.14 – VICE - modo de *Preview* de Impressão.

14.6.2 Aplicativos Java Desktop

Como destacado na seção de requisitos do Tdk, sua arquitetura deveria ser independente de linguagens. No caso de Java, as implementações necessárias para se ter o aplicativo descrito no exemplo anterior seriam bastante análogas ao caso de C++. No entanto, devido à impedância sintática entre as linguagens, devemos adicionar uma camada responsável pela troca de dados entre o mundo C++ e o mundo Java, conforme mostramos na Figura 14.15.

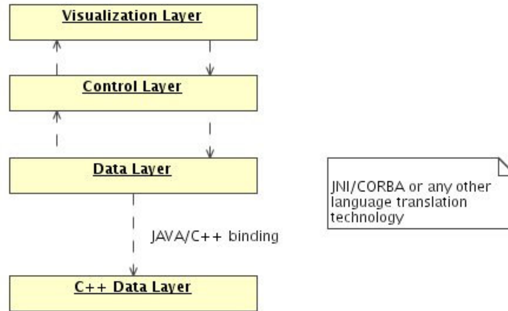

Figura 14.15 – A camada de acesso para abstração da troca de dados entre Java e C++.

14.6.3 Aplicativos Web

Um terceiro exemplo de produto desenvolvido com o Tdk é o aplicativo *Tdk Web Client* (ver Figura 14.16), que consiste em um visualizador de mapas na Web, compatível com o padrão WMS. Este aplicativo acessa o serviço WMS do Tdk (ver Seção 1.4).

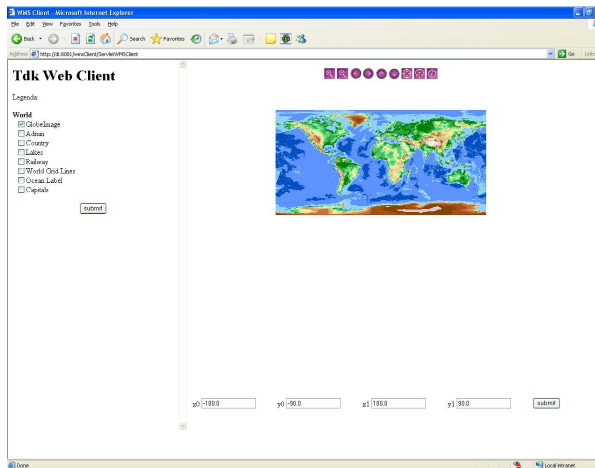

Figura 14.16 – Um produto Tdk de visualização de Mapas na Web.

Referências

GAMMA, E.; HELM, R.; JOHNSON, R.; VLÍSSIDES, J. Design Patterns - Elemens of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley,1995. 395 p.

GOLDBERG, A.; ROBSON. D. Smalltalk-80 : The Language and its Implementa-tion. Addison-Wesley, 1983.

KRASNER, E., G.; POPE, S. T.; A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, v. 1, n. 3, p. 26-49, 1988.

RUMBAUGH, J.; BLAHA, M.; PERMERLANI, W.; EDDY, F.; LORENSON, W.; *Object-Oriented Modeling and Design*. Prentice Hall , Englewood Cliffs , NJ, 1991.

SANTOS, A. L. S. C. VIX - Um Framework para Suporte a Objetos Visuais Interativos. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, 2005.