UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

UM CONVERSOR DE DADOS GEOGRÁFICOS, ABRANGENDO OS FORMATOS MID-MIF, SHP-DBF E SDL

ANDRÉ DEMBOSKI PINTER

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

UM CONVERSOR DE DADOS GEOGRÁFICOS, ABRANGENDO OS FORMATOS MID-MIF, SHP-DBF E SDL

AUTOR: ANDRÉ DEMBOSKI PINTER
ORIENTADOR: DR. ENG. ARMANDO LUIZ DETTMER
CO-ORIENTADOR: PROF. DR. RAUL SIDNEI WAZLAWICK
BANCA EXAMINADORA: PROF. DR. RONALDO DOS SANTOS MELLO

PALAVRAS-CHAVE: Sistemas de Informações Geográficas, SIG, GIS, Conversor, MapInfo, ESRI, MapGuide.

FLORIANÓPOLIS, 19 DE DEZEMBRO DE 2003

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

A monografia intitulada **UM CONVERSOR DE DADOS GEOGRÁFICOS, ABRANGENDO OS FORMATOS MID-MIF, SHP-DBF E SDL**, elaborada pelo acadêmico **ANDRÉ DEMBOSKI PINTER**, foi aprovada pela banca examinadora composta pelos professores abaixo subscritos, tendo sido julgada adequada para o cumprimento do requisito legal previsto no art. 9º da Portaria 1.886/94 do Ministério da Educação e Cultura - MEC, regulamentada pela Universidade Federal de Santa Catarina pela Resolução nº 003/95/CEPE.

Florianópolis, 19 de Dezembro de 2003.

_____
Dr. Eng. Armando Luiz Dettmer
Orientador

_____
Prof. Dr. Raul Sidnei Wazlawick
Co-Orientador

_____
Prof. Dr. Ronaldo dos Santos Mello
Membro da Banca

Dedico meus esforços a Deus; aos meus pais, Roseli e José Lourival, por serem meus exemplos de vida, e por tudo o que fizeram por mim; aos meus irmãos, Fernando e Rafael, pelo convívio familiar sempre fraterno, alegre e renovador; e aos meus amigos, por compartilharem dificuldades e alegrias nos mais diversos momentos de convivência.

Agradeço aos meus colegas do Laboratório de Transportes, por terem despertado em mim o interesse pela pesquisa; e ao meu orientador, Dr. Eng. Armando Luiz Dettmer, pela ajuda, paciência e amizade.

SUMÁRIO

LISTA DE FIGURAS

RESUMO

A monografia descreve os aspectos relacionados à conversão de diferentes tipos de dados geográficos, visando o compartilhamento de informações entre sistemas de informações geográficas distintos. Analisa e especifica os formatos abrangidos, como o MID-MIF (MapInfo), o SHP-DBF (ESRI) e o SDL (MapGuide), demonstrando os mecanismos envolvidos na conversão dos dados. A monografia apresenta ainda, uma descrição completa da implementação de um conversor multidirecional entre os três formatos citados, a ser utilizado pelo Laboratório de Transportes (ECV-UFSC) como uma ferramenta de troca de informações entre diversas bases geográficas.

Palavras-chave: Sistemas de Informações Geográficas, SIG, GIS, Conversor, MapInfo, ESRI, MapGuide.

ABSTRACT

This monography describes the aspects related to the conversion of different types of geographic data, in order to allow sharing of this information among distinct geographic information systems. It analyses and specifies the usual formats, like the MID-MIF (MapInfo), SHP-DBF (ESRI) and SDL (MapGuide), as well as the mechanisms involved in the data conversion process. It shows a complete description of the implementation of a multidirectional translator for the three formats, developed to be used by Laboratório de Transportes (ECV-UFSC) as an information interchange tool.

Keywords: Geographic Information Systems, SIG, GIS, Converter, Translator, MapInfo, ESRI, MapGuide.

1 INTRODUÇÃO

O intercâmbio de dados espaciais é um importante desafio no uso das geotecnologias, impulsionado principalmente pelo alto custo de produção deste tipo de dados. Em um ambiente de sistemas heterogêneos, a aquisição de dados representa um custo entre 60% e 80% do custo total na implantação de Sistemas de Informação Geográfica, segundo Hartman apud [LIMA et alli 2002].

Aliado a este fato, a maior parte dos produtos para suporte de SIG não são desenvolvidos para permitir a comunicação com outros sistemas computacionais. Genericamente, eles oferecem um ambiente próprio de trabalho e uma linguagem para interação com o sistema que, em alguns produtos, pode também ser utilizada na construção de procedimentos e desenvolvimento de aplicações.

Para modelar objetos e fenômenos georreferenciados, cada SIG utiliza um modelo conceitual próprio e esta diversidade faz com que organizações produtoras de informação georreferenciada sigam regras conceituais vinculadas ao sistema por elas utilizado. O resultado é um ambiente heterogêneo, onde cada organização tem sua maneira de tratar a informação espacial [LIMA et alli 2002].

Por vezes o uso de sistemas CAD pode trazer dificuldade no intercâmbio pois, em sua maioria, trabalham com dados muito genéricos, além de utilizarem um sistema de coordenadas próprio, desprezando as projeções cartográficas conhecidas. Outro fato é que muitos destes sistemas não respeitam o conceito de camada como nos SIGs, separando-as em arquivos diferentes para depois fazer um truncamento.

Unindo os problemas citados a outros não menos expressivos, podemos dizer que uma solução típica para resolvê-los consiste na tradução entre os dados disponíveis em geral para dados que possam ser manipulados pela aplicação desejada. Em geral, pode-se dizer que

este é o objetivo principal do trabalho: poder converter dados geográficos de formatos distintos, de forma bidirecional, a fim de torná-los compatíveis ao sistema que se deseja.

Desta forma, este trabalho visa realizar esta conversão através do desenvolvimento de um conversor de dados geográficos, inicialmente projetado para os formatos MID-MIF, SHP-DBF e SDL, mas que futuramente poderá ser estendido para outros formatos de interesse. Comercialmente, pode-se citar o software FME Suite 2004 (http://www.safe.com) como uma direção a ser buscada em caminhos futuros, já que este faz diversas conversões e manipulações, analisando perspectivas bastante abrangentes, porém com a desvantagem de não ser um produto *open source*.

O trabalho consiste então, primeiramente, de um embasamento teórico que dará uma visão geral dos sistemas de SIG, sua história e funcionalidades, detalhados no capítulo 2. Já o capítulo 3 trata da interoperabilidade desses sistemas, assim como as diversas abordagens existentes para esta interoperabilidade, apontando os pontos fortes e fracos de cada uma. O capítulo 4 vem para elucidar os formatos que serão utilizados pelo conversor, contendo uma breve descrição de cada um, como também referências para uma especificação mais detalhada. No capítulo 5 estão contidos os detalhes da implementação do conversor, sua estrutura interna e características. Por fim, o capítulo 6 apresenta as conclusões obtidas da elaboração do trabalho.

2 SISTEMAS DE INFORMAÇÃO GEOGRÁFICA (SIG)

2.1 Definição

Na era da informação como grande riqueza do nosso tempo, o nível de qualidade desta informação deve ser o mais substancial possível. Nas últimas décadas, diversas tecnologias apareceram para auxiliar na solução do problema acima. Entre estas, a aplicação da referência geográfica da informação em sistemas computacionais – os Sistemas de Informação Geográfica (SIGs) – possibilita dar uma melhor visualização do problema, facilitando a tomada de decisões [ALMEIDA 2002].

O termo SIG vem sendo objeto de várias definições por parte de diferentes autores. Porém, podemos dizer que fundamentalmente o termo SIG pode ser definido de duas maneiras distintas.

A primeira diz respeito à utilização para referir de forma genérica a um sistema de informação que contempla características relativas a localizações espaciais. Já a segunda utiliza o termo para referir um tipo determinado de produtos comerciais, especialmente direcionados para a realização de sistemas que envolvem dados representando localizações geográficas.

Entretanto, é importante distinguir o conceito de Sistema de Informação incluindo dados relativos a características espaciais de entidades georreferenciadas do de produto tecnológico voltado para a sua realização. Neste trabalho, opta-se por adotar o termo SIG mencionado em primeiro lugar, dado que focar-se-á na perspectiva de sua informatização. As ferramentas utilizadas no seu desenvolvimento serão consideradas apenas como uma das suas componentes.

2.2 História e Evolução

Efetivamente, a informação geográfica organizada por temas, tem sido tradicionalmente apresentada sob a forma de mapas desde as mais antigas civilizações. Recorrendo apenas a processos manuais, foi possível representar em folhas de papel o resultado das observações efetuadas sobre algumas características da superfície terrestre. Estas eram representadas por meio de pontos, linhas e áreas aos quais eram associados símbolos, cores e padrões, cujo significado era explicado numa legenda ou num texto. Com base neste tipo de mapas era possível realizar alguns tipos de análise. As primeiras operações de análise efetuadas tinham um caráter essencialmente qualitativo, já que se baseavam na mera observação visual e na intuição de quem efetuava a análise. Nos mapas baseados numa escala era também possível realizar algumas operações de análise quantitativa, basicamente relativas ao cálculo de distâncias e áreas [ABRANTES 1998].

No entanto, é perceptível que tais mapas apresentavam diversas limitações, devido ao caráter manual que lhes era inerente. Eles não eram ricos em detalhes, pois devido ao alto custo de produção adotavam-se escalas pequenas e, conseqüentemente a informação era representada com um alto nível de generalização. Além disso, encontravam-se desatualizados com rapidez, e é praticamente impraticável redesenhar a totalidade de um mapa sempre que a realidade é alterada. Finalmente, elaborar uma análise espacial sobre diversos temas era de elevada dificuldade na prática, sendo possível apenas para um volume pequeno de dados.

Desta forma, o recurso a meios computacionais para suporte de informação espacial iniciou-se no princípio da década de 60, com a codificação digital da informação que, tradicionalmente, apenas era representada sob a forma de mapas. Contudo, só os avanços no campo da tecnologia informática alcançados no início da década de 70, particularmente os relacionados com o acesso direto a discos, permitiram obter resultados significativos. Posteriormente, o enorme aumento de eficiência do processamento computacional permitiu o recurso a diversos tipos de análise espacial [ABRANTES 1998].

Inicialmente, os primeiros sistemas tinham o objetivo de produção automática de cartografia, relacionada a temas da responsabilidade de alguns organismos oficiais dos EUA. Entretanto, o CGIS (*Canada Geographic Information System*) é freqüentemente citado como sendo o primeiro SIG verdadeiro, pois além de produção de cartografia possibilitava também operações de análise espacial. A utilização deste tipo de sistemas, permitindo interpretar os dados segundo diferentes perspectivas, possibilitou uma visão melhor da informação, bem como formas novas de proceder e de apresentar resultados. O sucesso obtido deveu-se, em

grande parte, ao poder das representações no espaço para sugerir causas, explicações e relações.

Atualmente, os SIG são utilizados como ferramentas de análise geográfica, por excelência, já que permitem a integração de grandes volumes de informação espacial e de outros tipos num mesmo sistema e o seu tratamento conjunto. Esta integração tornou-se possível como resultado da convergência de várias disciplinas e técnicas tradicionais. Dentre estas podem citar-se como especialmente relevantes a Geografia, Cartografia, Fotogrametria, Detecção Remota, Agrimensão, Geodesia, Engenharia Civil, Matemática, Estatística, Investigação Operacional, Informática, e dentro desta as áreas de CAD (Computer Aidded Design), Computação Gráfica, SGBD (Sistemas Gestores de Bases de Dados), Redes e Inteligência Artificial [ABRANTES 1998].

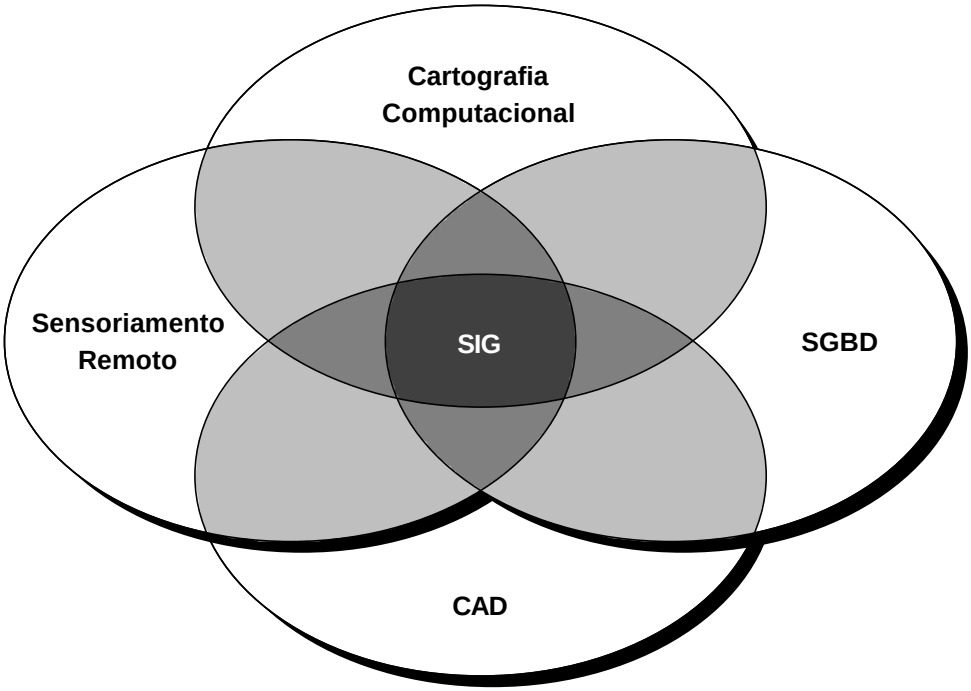

Figura 1 – Convergência das diversas áreas que resultaram no surgimento dos SIG.
Fonte: Adaptado de XAVIER

2.3 Funcionalidades

O termo Sistema de Informação Geográfica (SIG) é aplicado para sistemas que realizam o tratamento computacional de dados geográficos. Devido à sua ampla gama de

aplicações, que inclui temas como agricultura, florestas, cartografia, cadastro urbano e redes de concessionária (água, energia e telefonia), há pelo menos três formas de se utilizar um SIG:

- como ferramenta para produção de mapas;
- como suporte para análise espacial de fenômenos e
- como um banco de dados geográficos, com funções de armazenamento e recuperação de informação espacial [THOMÉ 1998].

Efetivamente, a diferenciação entre SIG e os outros sistemas como os de CAD ou de produção de cartografia toma como base a funcionalidade que esses sistemas disponibilizam. Neste âmbito, a distinção mais importante está no fato de que um SIG é capaz de realizar, automaticamente, a síntese de dados geográficos de diversas naturezas, incluindo os calculados pelo próprio sistema. Ou seja, a capacidade de produzir informações novas, que podem ainda ser usadas para atualizar dados desse mesmo SIG, é a qualidade que o distingue de outros sistemas de acesso simples a dados previamente registrados.

Pela abrangência expressa acima, observa-se que estes tipos de sistema lidam com informações multidisciplinares, onde a heterogeneidade e complexidade dos diversos temas é comum. Desta forma, os SIGs têm uma característica básica de integração de informações, tornando-se uma ferramenta que procura agregar dados artificialmente separados pelo homem, em sua tecnologia, de forma a manipulá-los e apresentá-los de outras maneiras, proporcionando uma nova visão ao usuário. Uma outra característica notada é a de suporte à decisão. Pela possibilidade de apresentar as informações existentes de uma outra maneira, através de manipulações e análise, provê ao usuário um suporte à tomada de decisão para melhor planejar o espaço em estudo, por exemplo [THOMÉ 1998].

Resumidamente, mesmo sabendo das diferentes definições de SIG, podemos dizer que elas expressam visões de diferentes pesquisadores, porém convergem para um mesmo ponto. Integrar em um único banco de dados informações espaciais de diferentes áreas, oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, e para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

3 INTEROPERABILIDADE ENTRE SIG

Segundo Almeida, os SIGs têm sido bastante utilizados a partir dos anos 80 como suporte para a tomada de decisões em áreas como administração pública, meio ambiente, marketing, etc. Cada produto de software comercial de SIG se desenvolveu independentemente, com poucas terminologias e teorias em comum. Como resultado, é bastante complicado para diferentes sistemas compartilharem dados, para usuários treinados em um sistema utilizarem outro, ou para compartilhar aplicativos desenvolvidos em diferentes sistemas. O termo interoperabilidade sugere um mundo ideal onde não existiriam estes problemas, ou pelo menos onde estes fossem minimizados.

Veremos então, a seguir, algumas abordagens para esta interoperabilidade aqui referida, como também suas características e problemas.

3.1 Conversão Sintática

A abordagem mais básica para a interoperabilidade entre SIGs é a conversão sintática direta, que faz a tradução de arquivos de dados geográficos de um formato para o outro. Para que seja permitido esse tipo de conversão os SIGs trabalham com duas alternativas:

- oferecer um formato de exportação ASCII de fácil legibilidade, como DXF (Autocad), MID/MIF (MapInfo), E00 (Arc/Info) e SPR (Spring);
- documentar as estruturas de dados internas, como no caso do SHP (ArcView).

Porém, a utilização destes formatos não exclui a possibilidade da ocorrência de distorções nos dados, quando da transferência das informações, pois eles são organizados de acordo com o sistema gerador e quando importados para sistemas conceitualmente diferentes, necessitam de manipulação externa.

Apesar das limitações da conversão sintática, deve-se reconhecer que a grande maioria dos processos de conversão de dados opera neste nível, assim como o conversor proposto por este trabalho, porém com algumas melhorias trazidas de outras abordagens.

Existem ainda formatos independentes de sistema para intercâmbio de dados como o SDTS - Spatial Data Transfer Standard (Estados Unidos), o SAIF – Spatial Archieve and Interchange Format (Canadá) e o NTF – National Transfer Format (Inglaterra).

O SDTS é projetado especificamente como um formato para transferência de dados, não para uso direto. Para isto, é especificado em partes que procuram abordar o nível conceitual, o lógico e o físico.

Apesar da decisão do Governo dos EUA em padronizar o SDTS para todos os órgãos federais americanos, este formato apresenta diversos problemas. O modelo conceitual do SDTS tem conteúdo semântico limitado, e está fortemente acoplado às definições do sistema Arc/Info-7. O padrão SDTS não contempla os conceitos de modelagem espacial orientada a objetos, não estabelece definições de metadados, não descreve formalmente relacionamentos espaciais e nem tem formas de capturar procedimentos de consulta e análise. Em resumo, o SDTS comporta-se como um formato de intercâmbio tradicional do tipo SHP ou MIF, sem grandes vantagens adicionais [LIMA et alli 2002].

3.2 Metadados

Metadados descrevem o conteúdo, condição, histórico, a localização e o formato do dado. O objetivo do seu uso é ter um mecanismo para identificar qual dado existe, a sua qualidade e como acessá-lo e usá-lo. A principal proposta de padrão de metadados é do FGDC (*Federal Geographic Data Committee*), comitê que promove a coordenação do uso, troca e disseminação de dados espaciais nos EUA. Este padrão ratifica um conjunto comum de definições para documentar o dado geográfico, incluindo: identificação, qualidade do dado, organização espacial do dado, referência espacial, informação sobre entidade e atributo, distribuição e referência do metadado [LIMA et alli 2002].

Outra ação do comitê FGDC é o patrocínio da criação de uma *clearinghouse* (National Geoespatial Data Clearinghouse), que se trata de um guia para fornecer aos usuários um melhor conjunto de dados espaciais para seus projetos, através de um site. Sendo isto feito através de pesquisa a metadados disponibilizados no padrão do FGDC por órgãos produtores de dados espaciais.

Como sua ênfase é na disponibilidade da informação, o padrão FGDC não especifica a maneira pela qual a informação está organizada nem o processo de transferência. Com exceção da parte de entidades e atributos, que pode revelar parte do significado do dado, as demais partes não descrevem a semântica da informação espacial.

Segundo Lima, o grande problema da proposta do FGDC (e do uso de metadados em geral) é a excessiva ênfase em informações que descrevem o processo de produção dos dados. Com relação à sintaxe, o padrão limita-se a indicar qual o formato em que os dados estão disponíveis. No aspecto semântico, suas informações são muito limitadas, pois o FGDC não adota o "modelo padrão" de geoinformação (campos e objetos). Adicionalmente, o padrão do FGDC reflete os compromissos inevitáveis do "projeto de comitê", pois requer uma excessiva quantidade de informações (de aplicação questionável), com dezenas de formulários.

Isto quer dizer que, o esforço desprendido para a elaboração da documentação e preenchimento dos formulários, exigidos por esse padrão, não compensa em razão do que é obtido como resultado.

Em resumo, a substancial burocracia envolvida em adotar o padrão FGDC não se traduz em benefícios proporcionais. Estes fatos talvez expliquem porque sua adoção ainda está limitada e porque o consórcio OpenGIS, sobre o qual será falado mais adiante, propõe seu próprio formato para metadados.

3.3 Conversão Semântica e Ontologias

O aspecto semântico diz respeito à representação conceitual da informação geográfica presente em cada sistema. Como comunidades com cultura e história diferentes que interpretam distintamente a realidade geográfica e produzem sistemas conceitualmente heterogêneos, a capacidade de transferir dados de um sistema para outro não garante que os dados têm significado para o novo usuário[LIMA et alli 2002].

Esta possível incompatibilidade aponta que, para que o intercâmbio aconteça, um conjunto consistente de interpretações deve estar disponível para a informação, levando a um significado comum sobre o dado trocado.

Analisando semanticamente, uma plena interoperabilidade entre diferentes formatos depende de compartilhamento de conceitos comuns entre os membros de uma comunidade de informação. Estão incluídos nestes conceitos: o modelo de dados; o dicionário de conceitos, indicado por uma ontologia comum; o dicionário de procedimentos, que contém

os diferentes mecanismos de consulta, manipulação e apresentação utilizados para extrair informação dos dados compartilhados.

Segundo Lima, o uso de Ontologias para interoperabilidade de dados geográficos ainda está nos seus primórdios, sem exemplos concretos nem padrões estabelecidos. Espera-se que, nos próximos anos, haja um substancial desenvolvimento neste campo.

3.4 XML

Além dos usuários de SIG, o problema da interoperabilidade atinge também usuários de outros sistemas computacionais, impulsionado principalmente pelo avanço das redes de computadores. O padrão XML surgiu, justamente, com o propósito de resolver este problema.

De acordo com a proposta de XML, o conteúdo de um documento (ou seus dados) é que devem ser codificados, mas não a sua forma de apresentação. A aplicação que utilizará estes dados é que deve ser responsável por interpretá-los e apresentá-los ao usuário.

O uso de XML é vantajoso, pois é ideal para descrever estruturas de dados hierárquicas e complexas, características comuns em dados espaciais. O dado permanece legível, pois usa a codificação ASCII, e acessível através de programas que acessam dados no padrão, os "*parsers* XML". Além destas vantagens, o uso de XML vem aumentando e este cada vez mais afirmado como padrão para armazenar e trocar dados [LIMA et alli 2002].

A utilização de XML para atuar no problema da interoperabilidade entre SIG foi uma proposta do consórcio OpenGIS, que resultou na publicação, em 2000, do *paper* de recomendação da Geographic Markup Language (GML) 1.0, baseada na tecnologia XML.

3.4.1 GML

GML foi especificada para o transporte e armazenamento de informação geográfica, incluindo propriedades espaciais e não espaciais das feições geográficas. Esta linguagem define, atualmente, três esquemas:

- *Feature.xsd* – Define tipos e elementos concretos e abstratos de acordo com a especificação OpenGIS;
- *Geometry.xsd* – Define a geometria de acordo com a especificação OpenGIS;
- *Xlinks.xsd* – Define formas de ligação entre documentos e elementos dentro de um documento XML.

Seguindo os princípios da linguagem em questão, estes esquemas podem ser estendidos para criar outros esquemas XML para aplicações específicas. Assim, prevalece o que a tecnologia XML propõe que é a possibilidade de estender os modelos, criando novas "*tags*", e cada instituição pode criar seus esquemas ou simplesmente utilizar esquemas de terceiros em seus SIGs, promovendo assim, a tão desejada interoperabilidade [SILVA 2002].

A utilização de esquemas XML oferece algumas facilidades na construção do código GML como, promover a utilização de elementos de esquemas diferentes para construir esquemas personalizados utilizando namespaces, permitir o controle sobre estruturas hierárquicas e possibilitar uma maior flexibilidade na definição de tipos de dados.

Dentre os principais objetivos da GML estão:

- Prover uma estrutura aberta para a definição de esquemas de aplicações e objetos espaciais;
- Permitir perfis que suportem subconjuntos das capacidades declarativas de GML;
- Suportar a descrição de esquemas de aplicações geoespaciais para domínios específicos e comunidades de informação;
- Permitir a criação e manutenção de esquemas e dados geoespaciais distribuídos e relacionados;
- Aumentar a habilidade das organizações para compartilhar dados geográficos;
- Servir como ferramenta para interoperabilidade de Sistemas de Informação Geográfica de uma maneira incremental [apud SILVA 2002].

A utilização de GML se baseia em conceitos que são comuns no contexto de SIG, como pontos linhas e polígonos e, por este fato, também apresenta deficiências nas questões semânticas.

Outro empecilho à sua utilização é a disponibilidade de ferramentas computacionais adequadas, uma vez que as ferramentas disponíveis não comportam as manipulações desejadas de dados e esquemas GML, ora permitindo só leitura, ora escrita. E quando leitura e escrita são permitidas não se permite manipulação de esquemas de aplicação.

3.5 O Padrão OpenGIS

A OGC (*Open GIS Consortium)* é uma organização internacional que está criando novas padronizações técnicas e comerciais para garantir interoperabilidade em SIG. Fundada em 1994 por fornecedores de software, companhias de telecomunicações, universidades,

provedores de informação e órgãos governamentais, entre outros, a OGC busca criar uma especificação de software e novas estratégias empresariais, a fim de tornar os sistemas de geoprocessamento abertos e integrar completamente os dados geográficos e as operações necessárias para manipulá-los.

Esta união de forças entre produtores de software, fabricantes de hardware, instituições de pesquisa e entidades governamentais é a alavanca que faz esta abordagem ganhar força no mercado em geral. Afinal, o fato de os diversos padrões de intercâmbio adotados em alguns países desenvolvidos não se firmarem como padrões mundiais é devido, certamente, ao caráter individual que possuem.

O padrão OpenGIS, está em fase de desenvolvimento e está tentando reunir esforços para que a grande heterogeneidade dos produtos de software de SIGs possam conversar entre si, e além disso, que as instituições possam trocar informações espaciais entre si, através de um embrião de uma proposta de Comunidade de Informação Espacial, que trata também de padronização dos procedimentos organizacionais das instituições [ALMEIDA].

Além disso, pretende definir um modelo de dados genérico e interfaces padronizadas para acesso a bancos de dados geográficos, baseadas em diferentes tecnologias, como XML, COM, Java e SQL. Esta abordagem segue o conceito de API (*Application Programming Interface*), que fornece uma forma unificada de acesso às funcionalidades de sistemas distintos.

Apesar de inegáveis avanços que a proposta OpenGIS apresenta no campo da interoperabilidade, ela ainda possui várias limitações. Entre elas, podemos citar que a existência de uma API resolve apenas o problema de acesso padronizado a bancos de dados espaciais e não substitui a necessidade da transferência dos dados entre sistemas. Isto se deve talvez ao fato desta proposta ter sido colocada em desenvolvimento com um certo atraso, o que fez com que SIGs com dados de estruturas diferentes se disseminassem pelo mercado, surgindo assim o problema da interoperabilidade.

Este fato, inclusive, é uma das motivações do desenvolvimento deste trabalho, pois se esta padronização tivesse sido colocada em prática antecipadamente, poderíamos ter sistemas de informações geográficas compatíveis, não havendo então a necessidade de conversores e tradutores.

Outro problema consiste no fato de que o OpenGIS pode ser visto como tendo menos poder expressivo do que diversos sistemas já atuantes no mercado, e seu mapeamento pode acarretar distorção significativa da informação transferida. Além disso, ele não permite a

definição de relacionamentos espaciais para a definição de restrições espaciais, como também possui problemas de especialização e hierarquia entre classes de objetos [LIMA et alli 2002].

3.6 Observações

Cientes das diversas abordagens de interoperabilidade e seus problemas, observa-se que um dos fatores que reconhecidamente impede uma maior difusão da tecnologia de geoprocessamento no Brasil é a falta de padrões nacionalmente estabelecidos para intercâmbio de dados geográficos.

Isso faz com que os usuários tenham que usar intensivamente formatos comerciais, definidos pelos softwares de SIG mais vendidos no mercado. Existem dois problemas principais relativos a esse procedimento. Em primeiro lugar, o formato definido por um determinado fabricante se aproxima mais das estruturas de dados usadas pelo seu produto do que das de qualquer outro produto, dificultando o intercâmbio. Além disso, os desenvolvedores de software comercial têm maior interesse na disponibilidade de diversos programas de importação e são menos exigentes quanto aos programas de exportação [MONTEIRO 2000].

Isto se deve à questão financeira envolvida, pois demandar esforços importando dados externos para a aplicação do desenvolvedor, para posteriores análises, é mais lucrativo do que exportar os dados dessa aplicação, para que concorrentes em potencial possam utilizá-los. Isto sem falar que a importação de dados é, geralmente, menos trabalhosa que a exportação, uma vez que extrai apenas as informações de utilidade à aplicação. Já a exportação requer um maior respeito aos padrões dos formatos, pois deve ser genérica para que qualquer aplicação que suporte tal formato possa importá-la.

Outro agravante que dificulta a adoção de padrões nacionais de intercâmbio de dados no país é o fato de que muitos formatos não apresentam documentação para consulta, ou documentação insuficiente. Desta forma, não se permite um bom estudo sobre tal formato em questão, impedindo muitas vezes o desenvolvimento de aplicações que possam interagir com esses dados.

4 FORMATOS DE INTERCÂMBIO DE DADOS GEOGRÁFICOS

Genericamente falando, pode-se dizer que a definição de padrões facilita o compartilhamento, a integração e a transferência de dados. Quando abordamos padrões para SIG, usualmente estamos nos referindo a padrões para linguagem de especificação, transferência de dados, geocodificação, documentação de metadados e de formatos.

Esses padrões vêm sendo definidos em diversos níveis, tais como internacional, nacional, federal, industrial ou qualquer outra forma de especificação aberta ao público. Alguns são legalmente estabelecidos (padrões *de jure*) enquanto outros são padrões adotados por consenso por uma parte significativa da comunidade (padrões *de facto*) [MONTEIRO 2000].

Tendo em vista o crescimento dos sistemas de informação geográfica disponíveis, vários formatos de intercâmbio foram surgindo para atender as necessidades dos clientes. Assim, as seções a seguir apresentam de forma resumida as principais características dos formatos abordados neste trabalho, bem como as referências para os interessados em maiores detalhes das respectivas especificações.

4.1 MID-MIF

O formato de intercâmbio MapInfo Interchange File (MIF) teve origem no lançamento da primeira versão do software MapInfo, pela empresa norte americana MapInfo Corporation, em 1994. Este formato de intercâmbio permite a leitura de arquivos MIF por outros SIG existentes [MONTEIRO 2000].

O MIF é um arquivo texto ASCII que descreve o conteúdo de uma tabela MapInfo e consiste, na verdade, de dois arquivos relacionados: um que armazena os dados gráficos e outro que armazena dados tabulares. Os dados gráficos estão em um arquivo com a extensão

.mif, enquanto os dados tabulares estão no outro arquivo com a extensão **.mid**. Estes arquivos podem ser lidos e escritos pelo software MapInfo Professional e traduzidos para outros formatos através de outros programas.

O arquivo MIF é composto por duas áreas – a área de cabeçalho do arquivo e a seção de dados – que seguem uma especificação bem detalhada para a sua construção. A seção de dados, que vem logo após o cabeçalho, pode conter diversas primitivas gráficas, uma para cada objeto gráfico. O MapInfo estabelece uma correspondência entre cada objeto especificado no arquivo MIF para cada linha de dado no arquivo MID, associando o primeiro objeto no arquivo MIF com a primeira linha no arquivo MID, o segundo objeto com a segunda linha e assim sucessivamente.

Quando da não existência de um objeto gráfico correspondente a uma linha particular no arquivo MID, um "objeto em branco" (NONE) deve ser escrito no arquivo MIF.

Desta forma, os seguintes objetos podem ser especificados:

- ponto (point)
- linha (line)
- poligonal (poly-line)
- região (region)
- arco (arc)
- texto (text)
- retângulo (rectangle)
- retângulo arredondado (rounded rectangle)
- elipse (ellipse)

O arquivo MID é um arquivo texto que tem seus dados delimitados de tal forma que cada linha corresponde a um registro de dados. Além disso, cada linha está associada a um objeto correspondente no arquivo MIF, que contém também a descrição dos campos em sua primeira parte. Caso o objeto de desenho no MIF não tenha um registro correspondente, deve ser colocada, no MID, uma linha com os campos em branco. O MID é um arquivo opcional, quando ele não é utilizado, todos os seus campos ficam em branco.

Além desta visão geral, mais detalhes da especificação do formato MID-MIF podem ser obtidas na referência [MAPINFO 1999], contida nas referências bibliográficas e também no site da MapInfo Corporation (www.mapinfo.com).

4.2 SHP – DBF

Um arquivo Shapefile (SHP) armazena, de forma não-topológica, características espaciais de geometria e atributos, em um conjunto de dados. As características de geometria são armazenadas num formato composto por um conjunto de coordenadas vetoriais.

Devido ao fato de não haver o excesso de processamento de uma estrutura de dados topológica, estes arquivos levam algumas vantagens em relação a outras fontes de dados, como velocidade de desenho e capacidade de edição mais rápidos. Além disso, os arquivos SHP geralmente necessitam de menos espaço em disco e são mais fáceis de ler e escrever.

Na verdade, um shapefile consiste de um arquivo principal, um arquivo de índice e uma tabela dBASE. O principal é um arquivo de registros variáveis, onde cada registro descreve uma informação geográfica, através da lista de seus vértices. No arquivo de índice, cada registro contém a posição do registro correspondente no arquivo principal, sempre em relação ao início do arquivo. Já a tabela dBASE contém características de atributos, sendo um registro por característica. Esta relação de um para um entre geometria e atributos é baseada no número de registros, ou seja, registros em arquivos dBASE devem estar na mesma ordem que os registros no arquivo principal.

Como forma de relacionar os arquivos que descrevem uma informação geográfica, devemos nomeá-los de forma igual, apenas diferenciando suas extensões. O arquivo principal, o arquivo de índice e a tabela dBASE possuem, respectivamente, extensões **.shp**, **.shx**, **.dbf**.

Neste formato, pode-se especificar os seguintes objetos:

- ponto (point)
- multiponto (multipoint)
- poligonal (polyline)
- polígono (polygon)

Um simples arquivo shp pode conter diversos objetos de desenho, porém, todos eles têm que ser do mesmo tipo (arquivo só de linhas, ou só de pontos, etc).

Informações mais detalhadas, além desta breve descrição, podem ser encontradas na descrição técnica da ESRI que está referenciada na bibliografia [ESRI 1998], e também no site da empresa (www.esri.com).

4.3 SDL

O formato Spatial Data Load (SDL) consiste de um arquivo no formato ASCII utilizado pelo software Autodesk MapGuide e também por outras ferramentas de manipulação de mapas na World Wide Web.

A representação dos dados em SDL pode ser feita tanto em duas dimensões como também em três dimensões (2D e 3D). Este tipo de arquivo armazena ambas as informações, de geometria e de atribuição. Em tempo, um conjunto lógico de dados em SDL consiste de um ou mais arquivos contidos em um mesmo diretório, todos eles possuindo a extensão **.sdl**.

As representações geométricas podem ser dos seguintes tipos:

- ponto (point)
- linha (line)
- poligonal (polyline)
- polígono (polygon)

Entretanto, cada arquivo SDL deve conter apenas um tipo geométrico. É permitido também que sejam armazenadas informações sem características geométricas. Além disso, os tipos de objetos podem ser múltiplos. Desta forma, um objeto ponto pode ser formado, na verdade, por vários pontos, um objeto linha ser formado por várias linhas e assim por diante.

O SDL é um formato muito fechado, onde é muito difícil obter informações a respeito de sua estrutura e características tanto na Internet como em outras fontes de pesquisa. Além da bibliografia contida nas referências [SAFE 2004], talvez seja possível obter algumas informações interessantes no site da Autodesk (http://usa.autodesk.com).

5 CONVERSOR DE DADOS GEOGRÁFICOS

O conversor de dados geográficos desenvolvido neste trabalho permite o intercâmbio dos formatos MID-MIF, SHP-DBF e SDL, minimizando o máximo possível as distorções, a fim de que sejam imperceptíveis ao usuário de SIG. Adicionalmente, este será de suma importância para o Laboratório de Transportes, uma vez que possibilitará a conversão de diversas bases de dados disponíveis na Internet para os formatos utilizados por suas aplicações.

A implementação foi feita através da ferramenta de desenvolvimento Borland Delphi 7, em ambiente Windows XP.

Internamente, a estrutura do conversor consiste de diversos parsers, um para cada formato de arquivo, que têm a responsabilidade de manipular os dados de seus respectivos formatos. Desta forma, a extensibilidade do conversor para a manipulação de outros formatos geográficos se torna mais fácil, pois basta, praticamente, desenvolver o parser específico para um novo formato em questão.

5.1 Componentes Utilizados

Para o desenvolvimento do conversor, alguns componentes extras foram utilizados juntamente com a ferramenta de desenvolvimento e, a instalação destes, é imprescindível para compilar-se o seu código fonte. Os componentes utilizados foram os da biblioteca RxLib, desenvolvido pelo projeto JEDI, que pode ser encontrado em: http://homepages.borland.com/jkaster/ccds/delphi7disk1/project_jedi/rxlib_2_75_for_d7_from_project_jedi/.

5.2 Compatibilidade dos Objetos de Desenho

Como foi visto anteriormente, no capítulo 4, tanto o MID-MIF, quanto o SHP-DBF e o SDL possuem objetos de desenho particulares, sendo diferentes de formato para formato. Assim, para que ocorra uma conversão de um tipo de dado para um outro de estrutura diferente, é necessário que haja um mapeamento, respeitando o formato de destino.

No conversor, este mapeamento foi realizado de acordo com a seguinte tabela:

Tipos de Objeto	Interno	MIF	SHP	SDL
Ponto	TCustomNode	Point	Point	Point
Linha	TCustomLine	Line	Polyline	Line
Poligonal	TCustomLine	Polyline	Polyline	Polyline
Região	TCustomArea	Region	Polygon	Polygon
Arco	*	Arc	-	-
Texto	*	Text	-	-
Retângulo	TCustomArea	Rectangle	Polygon	Polygon
Retângulo arredondado	TCustomArea	Rounded rectangle	-	-
Elipse	-	Ellipse	-	-

***** Em implementação
- Objeto não suportado

Tabela 1 – Mapeamento dos objetos de desenho entre os formatos abrangidos pelo conversor.

Para uma melhor compreensão da tabela, pode-se exemplificar uma conversão hipotética: Um usuário, possuindo um arquivo MIF de linhas, deseja convertê-lo para o formato SHP. Desta forma, o parser de MIF se encarrega da leitura do arquivo inicial, as *lines*, e do mapeamento para o formato interno do conversor, que neste caso também são *lines*. Tendo feito isso, o parser de SHP se encarrega de, a partir no formato interno, fazer o mapeamento para *polylines*, armazenando os dados manipulados em um arquivo diferente.

Esta lógica é estendida para todos os outros objetos de desenho fazendo com que, desta forma, o conversor respeite as particularidades de cada formato.

5.3 Estrutura

Como mencionado anteriormente, o conversor é composto por diversos parsers, responsáveis por manipular os dados do formato a que eles se referem. Assim, podemos dizer que os parsers possuem, de forma genérica, estruturas bem semelhantes, pois devem ler os dados de seu formato e convertê-los para o formato interno, assim como também a operação inversa, receber dados no formato interno e convertê-los para seu formato. A estrutura do funcionamento de um parser pode ser melhor visualizada na figura 2:

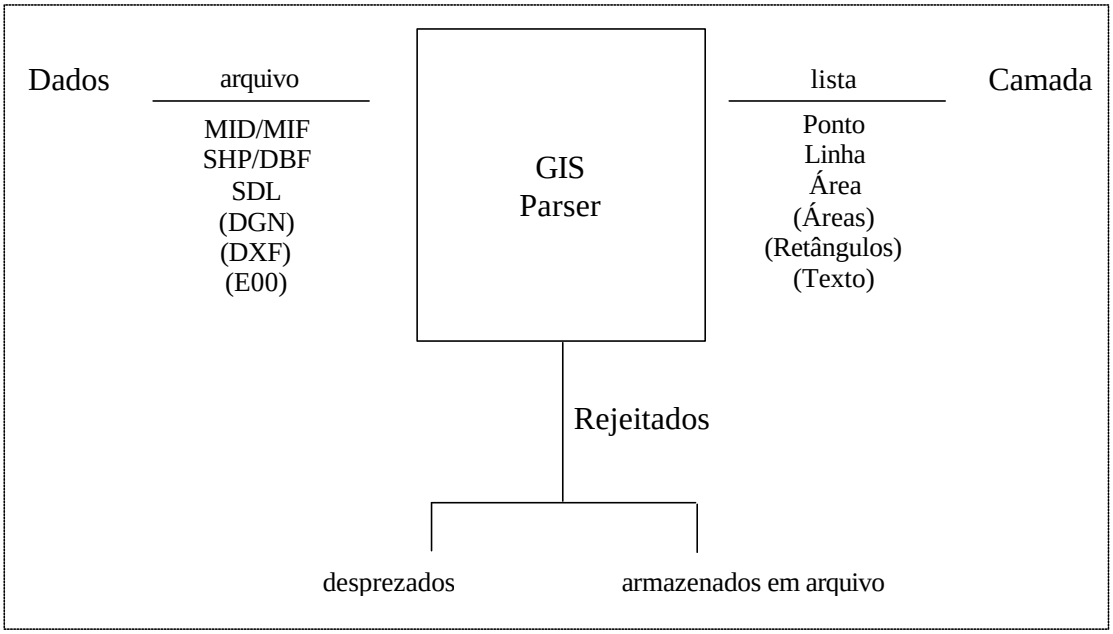

Figura 2 – Modelo genérico do funcionamento de um Parser. (As palavras entre parênteses significam formatos a serem suportados futuramente).

Aprofundando-se ainda mais internamente na estrutura do parser, percebemos que a conversão é executada por etapas, como pode ser observado na figura 3.

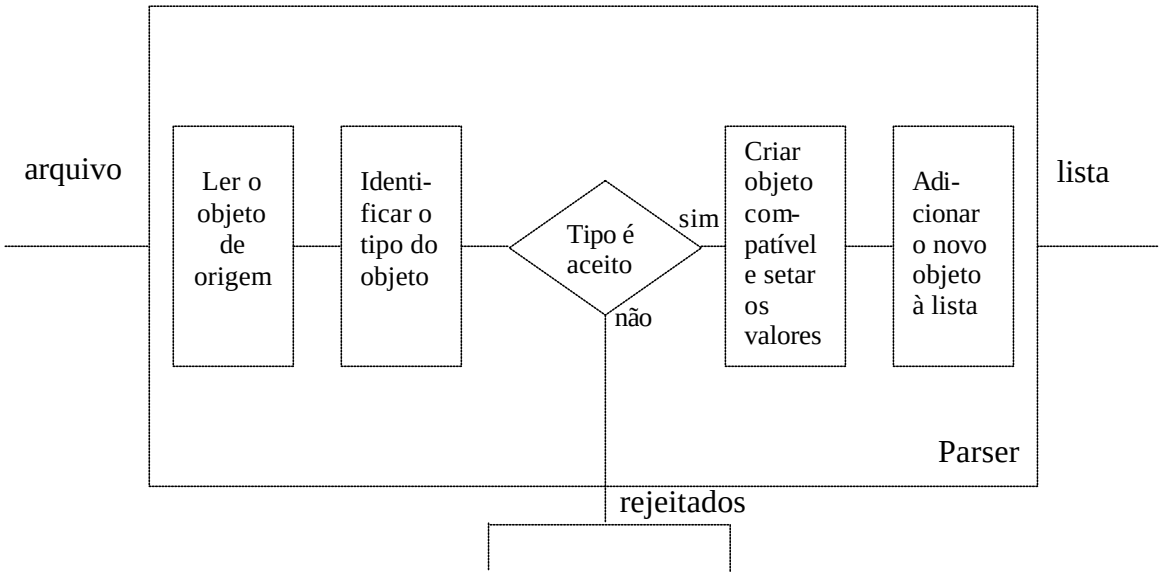

Figura 3 – Tratamento dos objetos de desenho pelo respectivo Parser.

Observa-se também, nesta figura, que o objeto lido a partir do arquivo de origem pode não ter correspondência no formato de destino. Entretanto, o conversor oferece ao usuário a possibilidade de salvar estes objetos em um arquivo em separado, ou então, descartá-los definitivamente.

Em um processo de conversão, os objetos de desenho são lidos a partir do arquivo de origem, pelo parser do formato a qual este arquivo pertence. No momento desta leitura, cada objeto é identificado (ponto, linha, área, etc...) e é conferida a sua possibilidade de mapeamento para o formato interno do conversor. Em caso positivo, cria-se uma nova instância do objeto interno adicionando-o como elemento de uma camada, sendo este processo repetido até que todos os objetos do arquivo tenham sido inseridos na camada. Após a inserção de todos os objetos contidos no arquivo, esta camada, que também foi criada em tempo de execução, é adicionada a uma lista de camadas, que é interna ao conversor.

Desta forma, para cada arquivo de origem, é criada uma camada interna, e a partir de cada camada, pode-se fazer o processo inverso, gerando-se arquivos de destino no formato desejado.

Como se pôde observar, a estratégia utilizada é a de transformar todos os arquivos de origem, dos mais diversos formatos, em camadas internas e, a partir delas, transformá-las em arquivos do formato de destino escolhido. Desta forma, simplifica-se o processo, pois os

parsers precisam apenas interagir entre o seu formato e o interno e não entre todos os formatos abrangidos pelo conversor.

5.4 Diagrama de Classes

O diagrama de classes do conversor, apresentado neste trabalho, expõe apenas as classes e seus relacionamentos e associações. O detalhamento destas classes, como atributos e métodos, serão especificados mais adiante.

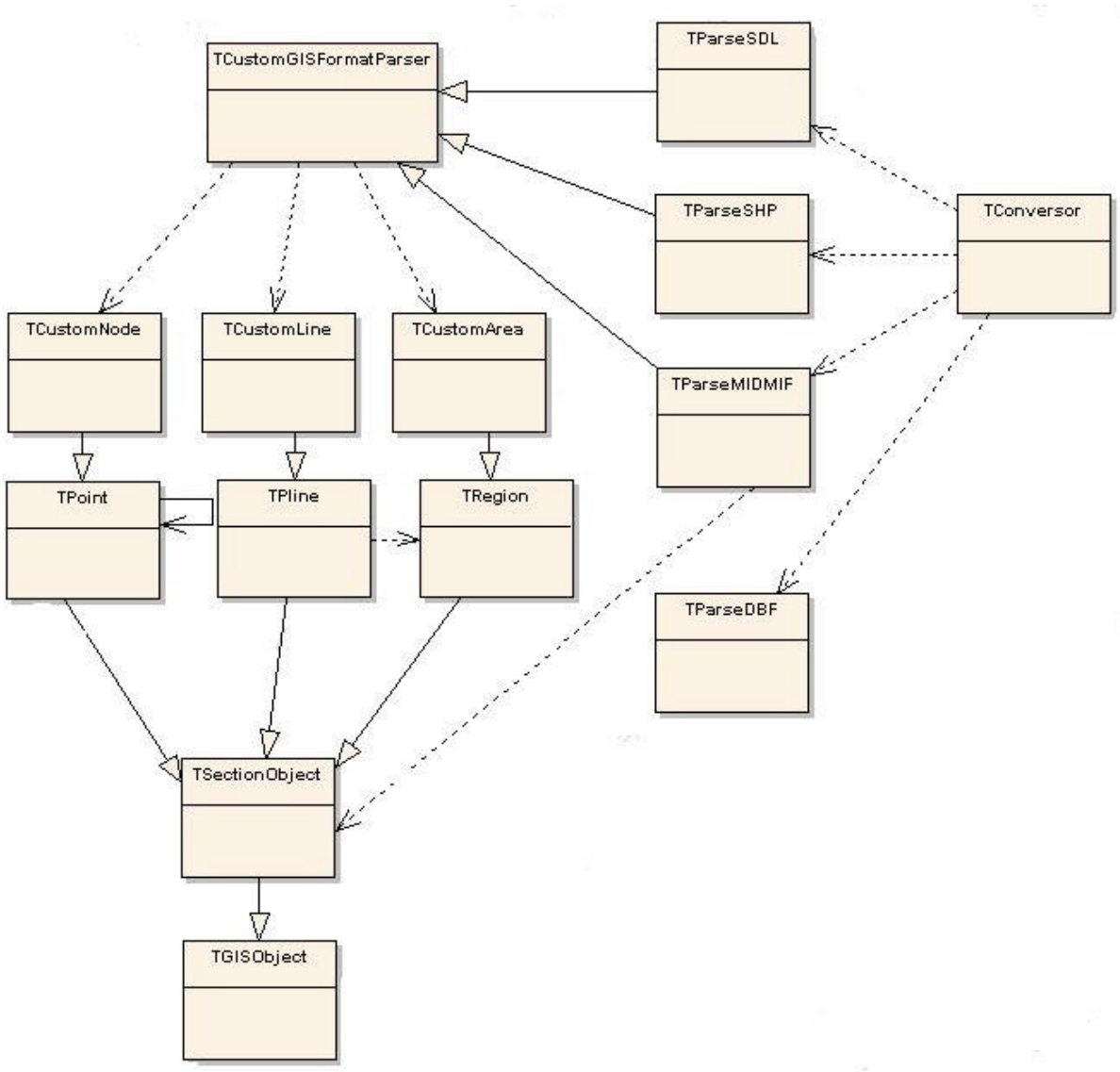

Figura 4 – Diagrama de classes do Conversor de Dados Geográficos.

5.5 Descrição das Classes

Para efeito de esclarecimento, só serão explicadas nesta seção as classes mais genéricas, sendo que suas especializações serão apenas citadas.

Em tempo, a classe TConversor serve como um gerente das outras classes, o objeto desta classe é que invoca os métodos dos outros objetos, caminhando para a solução do problema.

5.5.1 TCustomGISFormatParser

Esta classe representa uma entidade "parser", possuindo atributos e métodos para manipular os objetos de desenho, assim como fazer a leitura e escrita desses objetos. Seus descendentes são responsáveis por fazer esta manipulação de acordo com o tipo que se referem. Por exemplo, a TParseSDL manipula dados de arquivos **.sdl**, TParseSHP de arquivos **.shp**, e assim por diante.

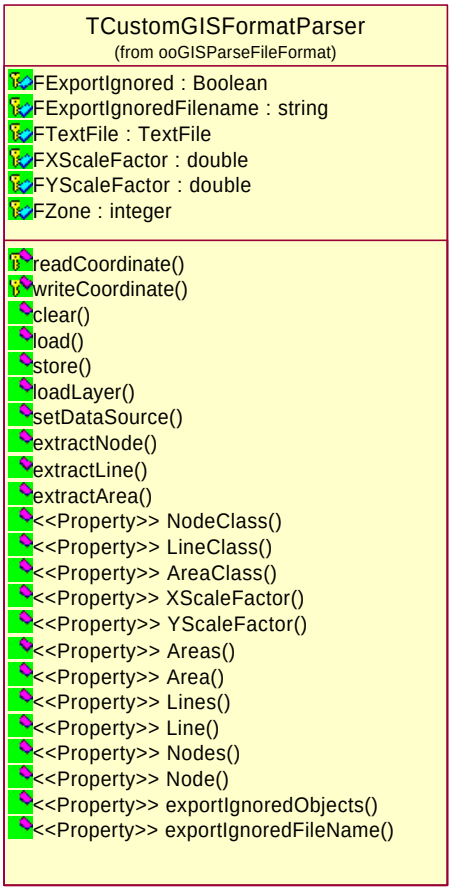

Figura 5 – Classe TCustomGISFormatParser.

5.5.2 TParseDBF

Apesar de esta classe também representar um “parser”, ela é apresentada em separado devido ao fato de não descender de TCustomGISFormatParser, pois um arquivo **.dbf** não é um arquivo de formato de SIG. Porém, sua funcionalidade é muito semelhante às outras classes de “parser”, manipular os dados de arquivos a qual se refere.

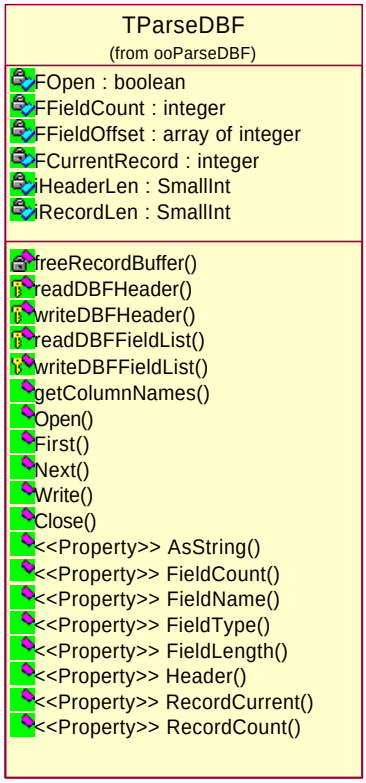

Figura 6 – Classe TParseDBF.

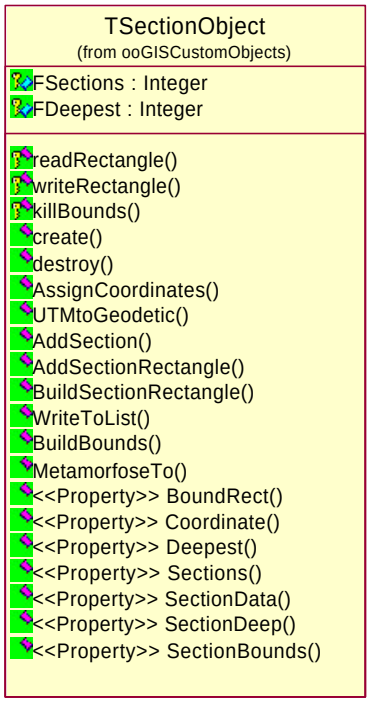

Figura 7 – Classe TSectionObject.

5.5.3 TSectionObject

Esta classe representa os objetos de desenho que podem ser múltiplos. Sua superclasse é a TGISObject, que possui as características básicas de um objeto de desenho. As especializações desta classe servem para representar os três tipos básicos dos objetos de desenho internos.

5.6 Interface

5.6.1 Tela inicial de origem dos arquivos

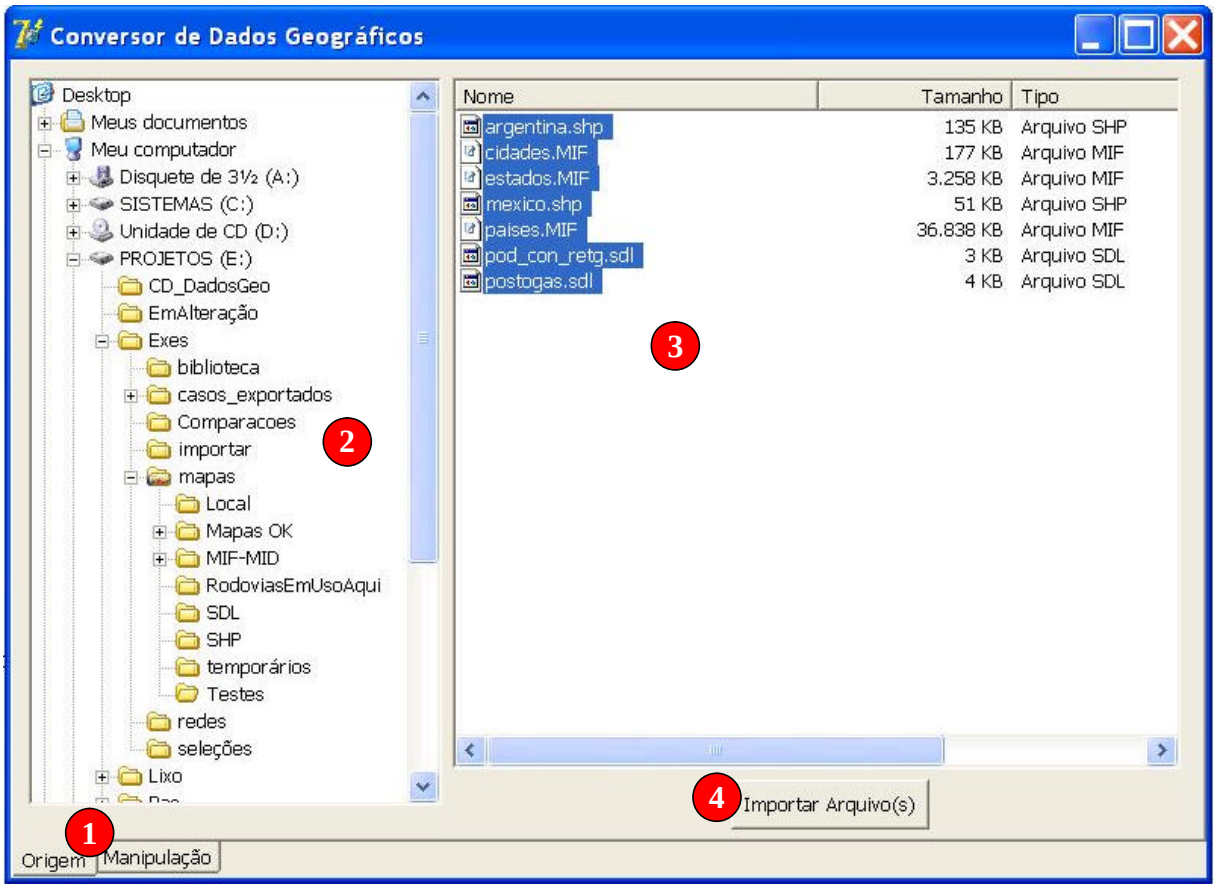

Figura 8 – Tela inicial do conversor.

Em “1” tem-se o *control page* que serve para alternar entre a página dos arquivos de origem a serem convertidos (fig. 8) e a página de manipulação da conversão (fig. 9).

Já em “2”, observa-se a árvore de diretórios da máquina do usuário, onde deve-se selecionar o diretório que contém os arquivos a serem convertidos pro formato desejado.

Em “3”, deve-se selecionar os arquivos que deseja-se converter e adicioná-los à lista dos arquivos para conversão, pressionando “4”.

Desta forma, pode-se ir em busca de mais arquivos a serem convertidos, sempre lembrando de adicioná-los utilizando “4”.

Em seguida, após todos os arquivos a serem convertidos terem sido importados, deve-se utilizar “1” para alternar para a página de manipulação, vista na subseção seguinte.

5.6.2 Tela de manipulação

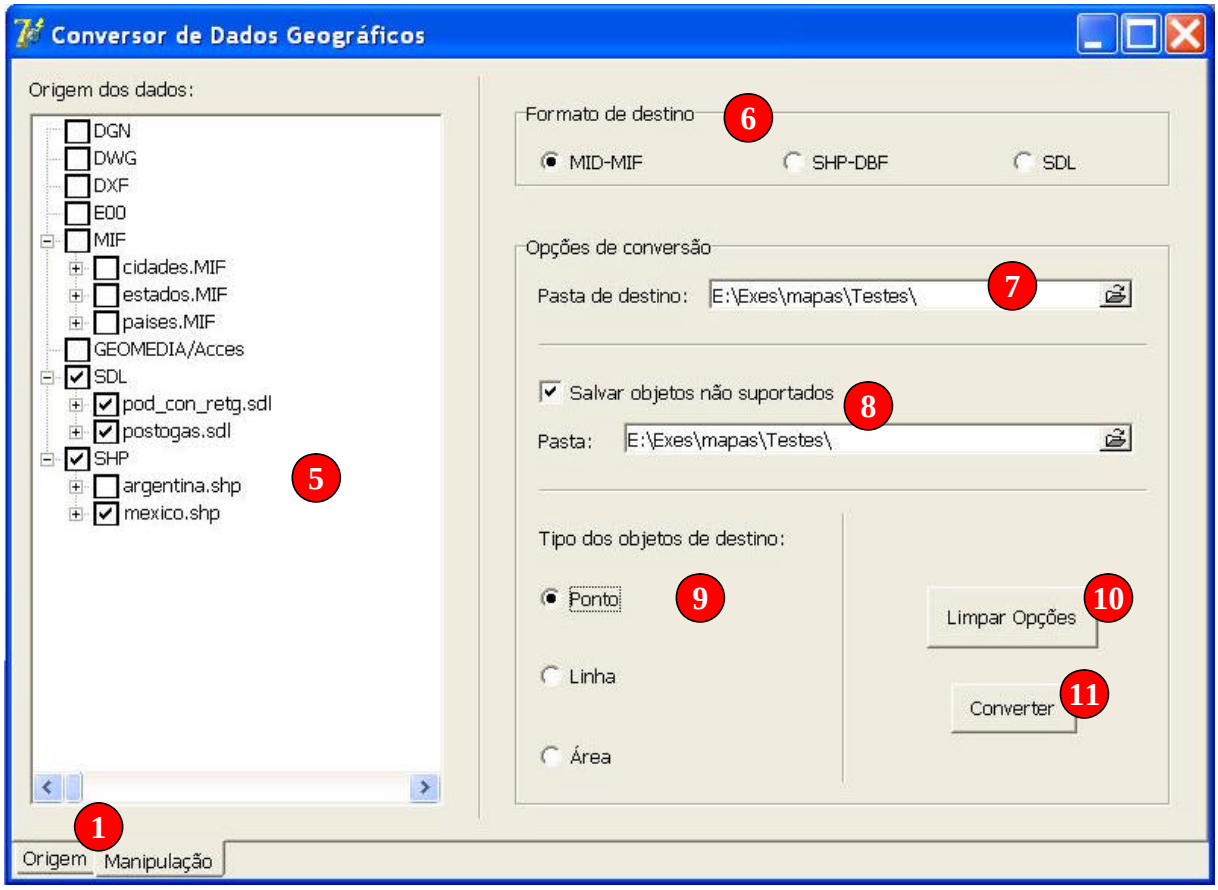

Figura 9 – Tela de manipulação dos dados, no conversor.

Em “5” pode ser observado a árvore de formatos do conversor, sendo que cada nó principal corresponde a um formato de arquivo suportado. Entretanto, prevendo futuras conversões, a árvore já possui formatos ainda não suportados por esta versão do conversor. Nesta árvore, deve-se selecionar os arquivos que deseja-se converter, tomando o cuidado de escolher arquivos que representem objetos geográficos do mesmo tipo, como só pontos, só linhas e assim por diante.

O formato de destino deve ser escolhido em “6”, selecionando-se o *radio buttom* correspondente.

Em “7”, deve-se escolher o diretório em que serão gerados os arquivos de destino.

Quando há impossibilidade de mapeamento entre os objetos de desenhos dos formatos, o conversor pode descartar esses objetos ou armazená-los em outro arquivo, e esta opção deve ser feita em “8”. Caso opte-se por armazenar, deve-se escolher uma pasta de destino.

Em “9”, seleciona-se o tipo dos objetos geográficos que devem ser lidos dos arquivos de origem, sendo que, tendo o arquivo de origem mais de um tipo de objeto de desenho, só os do tipo especificado neste local serão importados para o formato interno, descartando-se os outros.

O botão referenciado por “10” serve para limpar as opções de manipulação, retornando seus campos aos valores iniciais.

Finalmente, em “11”, está o botão que executa todo o processo de conversão, desde a leitura dos arquivos de origem até a geração dos arquivos de destino.

6 CONCLUSÃO

Durante a elaboração do trabalho, observou-se que o termo Sistemas de Informação Geográfica (SIG) pode ter diversas definições, de acordo com os diferentes autores da comunidade científica. Entretanto, o foco de sua aplicação é um fator que está bem claro e definido, que é integrar as informações em um único banco de dados e oferecer mecanismos de análise e manipulação sobre elas.

Devido a sua aplicação em diversas áreas, científicas e comerciais, os SIG desenvolveram-se em separado, atendendo os desejos de cada área, e este é um dos fatores que impedem que estes sistemas interajam entre si atualmente. Mesmo com as diversas abordagens de interoperabilidade citadas no trabalho, percebeu-se que fazer esta integração é uma tarefa muito difícil, e que certas vezes acarreta em perda de informação para o usuário. Ou seja, o desenvolvimento de formatos proprietários dificulta o livre intercâmbio de informação, levando a comunidade a preferir padrões neutros de intercâmbio.

No Brasil, isso não é diferente, a falta de padrões nacionais para intercâmbio de dados geográficos faz com que os usuários tenham de usar formatos comerciais, dificultando ainda mais a interoperabilidade, uma vez que cada formato comercial possui estruturas de dados próprias e, muitas vezes, sem documentação.

Analisando o intuito do conversor de dados desenvolvido neste trabalho, pode-se afirmar que ele auxilia a integração de SIGs distintos, pois permite fazer as conversões de dados necessárias para que estes sistemas interajam. Este auxílio se dá principalmente ao Laboratório de Transportes que, de posse do conversor, pode agora disponibilizar às suas aplicações, dados geográficos antigamente não importados por elas.

Resultados obtidos em conversões entre os formatos MIF e SHP foram satisfatórios, pois apresentaram um alto percentual de acerto, não apresentando qualquer

deficiência. Com o formato SDL, encontrou-se alguma dificuldade, até mesmo pela falta de documentação deste formato.

Entretanto, ainda há um longo caminho a ser percorrido para que o conversor possa solver problemas de interoperabilidade no âmbito geral, encobrindo diversos formatos nas diversas áreas de interesse. Desta forma, para trabalhos futuros, sugere-se a ampliação dos formatos abrangidos, assim como também manipulações mais avançadas sobre estes dados. Como citado anteriormente, o software FME Suite 2004 (http://www.safe.com) pode ser mencionado como um objetivo a ser buscado, já que ele permite manipulações de dados como a composição de um arquivo de destino a partir de vários arquivos de origem, sendo estes do mesmo formato ou não, além de outras manipulações de alta complexidade.

7 REFERÊNCIAS BIBLIOGRÁFICAS

ABRANTES, G. Sistemas de Informação Geográfica – Conceitos. 1998. Disponível em: http://www.isa.utl.pt/dm/sig/sig/SIGconceitos.html. Acesso em: 17 dez. 2003.

ALMEIDA, M. A. Análise da interoperabilidade em sistemas de informação geográfica. **Ratio: revista do Instituto Luterano de Ensino Superior de Ji-Paraná** – Revista da Faculdade ULBRA. Ji-Paraná – MT, n. 5, p. 3-9. Jun, 2002.

ESRI. ESRI Shape File Technical Description. Jul, 1998. Disponível em: www.esri.com/library/whitepapers/pdfs/shapefile.pdf. Acesso em: 18 dez 2003.

LIMA, P.; CÂMARA, G.; QUEIROZ, G. GeoBR: Intercâmbio Sintático e Semântico de Dados Espaciais. In: Simpósio Brasileiro de Geoinformática. IV. 2002. Caxambu – MG. Anais GEOINFO 2002. P. 139 – 146.

MAPINFO COMPORATION. The MapInfo Interchange File (MIF) Format Specification. Out, 1999. Disponível em: http://ees2.geo.rpi.edu/gis/data/mapinfo_mif.pdf. Acesso em: 18 dez 2003.

MONTEIRO, R. P. **Intercâmbio de dados entre Sistemas de Informação Geográficos.** 2000. Monografia (Especialização em Informática Pública). PUC – MG.

SAFE SOFTWARE. Feature Manipulation Engine (FME) Readers and Writers. Cap. 56. Disponível em: ftp://ftp.safe.com/fme/2004/docs/ReadersWriters2004.zip. Acesso em: 23 jan 2004.

SILVA, P. P. O. da. **Codificação XML para armazenamento e transporte de dados geográficos.** 2002. Trabalho acadêmico (Mestrado em Informática). UFRJ. Rio de Janeiro – RJ.

THOMÉ, R. **Interoperabilidade em geoprocessamento: conversão entre modelos conceituais de Sistemas de Informação Geográfica e comparação com o padrão OpenGIS.** 1998. Dissertação (Mestrado em Computação Aplicada). INPE. São José dos Campos – SP.

XAVIER, C. C. Sistemas de Informação Geográficas – Trabalhos em desenvolvimento pelo IMPA e INPE. Disponível em: http://orion.lcg.ufrj.br/seminarios/gis.ppt. Acesso em: 28 dez 2003.

APÊNDICE – Exemplos dos Formatos

MIF

```
Version 300
Delimiter ";"
CoordSys Earth Projection 1, 0
Columns 9
AREA Float
CODE Char(4)
NAME Char(25)
POP1990 Integer
POP90_SQMI Float
P_URBAN90 Float
P_ING_LANG Float
P_EMPL_SEC Float
HSE_UNIT90 Integer
DATA

REGION 3
 3
-113.13971700 29.01777600
-113.24057000 29.06777500
-113.45084300 29.28666300
 5
-115.17945800 28.02471900
-115.30388600 28.09888800
-115.35529300 28.09027400
-115.24973200 28.22833200
-115.28056300 28.31583000
 4
-115.01789800 31.94748800
-115.03527800 31.95722100
-115.01445000 31.90777500
-114.95305600 31.89638500

  CENTER 0.00000000 0.00000000


REGION 1
 6
-105.40209900 23.06745900
-105.46279900 23.04083000
-105.48780000 22.97110900
-105.44419800 22.90389000
-105.54170200 22.83694000
-105.57530200 22.75304900

  CENTER 0.00000000 0.00000000
```

MID

```
28002.325,"MX02","Baja California Norte",1660855,61.537060,90.904620,
1.300000,31.700000,362727

27898.191,"MX03","Baja California Sur",317764,11.204120,78.254620,
1.000000,18.800000,67304

10547.762,"MX18","Nayarit",824643,79.186840,62.054850,3.400000,17.600000,
168451

30736.386,"MX14","Jalisco",5302689,169.943300,81.853420,0.500000,32.700000,
1029178

2110.761,"MX01","Aguascalientes",719659,340.778900,76.521930,0.100000,
34.200000,129853
```

SHP

Este formato não pode ser representado como texto, pois não é um formato ASCII. Porém possui a seguinte estrutura:

- Cabeçalho do arquivo: 100 bytes.
- Cabeçalho do objeto: 12 bytes.
- Coordenadas do objeto: depende do objeto (ponto, linha, etc).

Repete-se, desta forma, os dois últimos itens, tantas vezes quanto forem os objetos de desenho.

DBF

Este arquivo também não pode ser representado como texto, ele tem o formato de uma tabela dBASE.

SHX

Este também não é um arquivo ASCII. Ele armazena a posição de início de cada objeto de desenho no arquivo SHP, sendo esta posição sempre marcada a partir do início do arquivo.

SDL

```
M, "TARCISO DE AGUIAR", "1700033920", 1
6851704.35, 699556.77
M, "AUTO POSTO VIANA", "1700040532", 1
6851473.23, 699476.5
```

ANEXO 1 – Código Fonte

Devido ao elevado número de linhas de código, são apresentadas aqui apenas as principais “units”, sendo que os fontes completos estão anexados em mídia.

```
program CDG;

uses
  Forms,
  MainForm in 'MainForm.pas' {Conversor},
  ooGIS in 'ooGIS.pas',
  ooGISParseBinaryGIS in 'Parsers\ooGISParseBinaryGIS.pas',
  ooGISParseFileFormat in 'Parsers\ooGISParseFileFormat.pas',
  ooGISParseMidMif in 'Parsers\ooGISParseMidMif.pas',
  ooGISParseSDL in 'Parsers\ooGISParseSDL.pas',
  ooGISParseSHP in 'Parsers\ooGISParseSHP.pas',
  ooParseDBF in 'Parsers\ooParseDBF.pas',
  ooGISCustomObjects in 'ooGISCustomObjects.pas',
  ooGISLayers in 'ooGISLayers.pas',
  ooGISAttributes in 'ooGISAttributes.pas';

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TConversor, Conversor);
  Application.Run;
end.
```

```
unit MainForm;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ComCtrls, StdCtrls, ShellCtrls, TreeViewEx, ToolWin, ActnMan,
  ActnCtrls, ExtCtrls, Mask, ToolEdit, CheckLst,Contnrs,ooGIS;

type
  TConversor = class(TForm)
    PageControl1: TPageControl;
    TabSheet1: TTabSheet;
    TabSheet2: TTabSheet;
    stv_Files: TShellTreeView;
    slv_Files: TShellListView;
    tv_Formats: TTreeViewEx;
    Label1: TLabel;
    Bevel1: TBevel;
    gb_Format: TGroupBox;
    rb_Midmif: TRadioButton;
    rb_Shpdbf: TRadioButton;
    rb_Sdl: TRadioButton;
    gb_Opcoes: TGroupBox;
    Label2: TLabel;
    cb_SaveUnsuported: TCheckBox;
    Bevel2: TBevel;
    Label3: TLabel;
    Bevel3: TBevel;
    Button1: TButton;
    Panel1: TPanel;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    RadioButton3: TRadioButton;
    Label4: TLabel;
    Bevel4: TBevel;
    fe_FileDestiny: TDirectoryEdit;
    DirectoryEdit1: TDirectoryEdit;
    Button2: TButton;
    btn_Importar: TButton;
    procedure slv_FilesAddFolder(Sender: TObject;
      AFolder: TShellFolder; var CanAdd: Boolean);
    procedure tb_ImportarClick(Sender: TObject);
    procedure PageControl1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure cb_SaveUnsuportedClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure RadioButton1Click(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure fe_FileDestinyExit(Sender: TObject);
    procedure Button2Click(Sender: TObject);

    procedure setScaleUP;
    procedure setScaleDown;
  private
    { Private declarations }
    nodeMIF :TTreeNode;
    nodeSDL :TTreeNode;
    nodeSHP :TTreeNode;

    LayerList      :TStringList;
    DestinyObjKind :TObjectKind;

  protected
    procedure insertFile(aPath :string);
    procedure loadFromFormatFiles;
    procedure storeIntoFormatFiles;
```

```
    procedure ObjKindClick;
    procedure clearFormatTree;

  public
    { Public declarations }
  end;

var
  Conversor: TConversor;

implementation

uses ooGISParseMidMif,
     ooGISParseSHP,
     ooParseDBF,
     ooGISParseSDL,
     ooGISCustomObjects;
{$R *.dfm}

procedure TConversor.slv_FilesAddFolder(Sender: TObject;
  AFolder: TShellFolder; var CanAdd: Boolean);
var ext :string;
begin
  ext:= upperCase(ExtractFileExt(AFolder.PathName));
  CanAdd:= {(ext = '.MID') or} (ext = '.MIF') or
           (ext = '.SHP') or {(ext = '.DBF') or (ext = '.SHX') or}
           (ext = '.SDL') or (ext = '.MDB') or
           (ext = '.DGN') or (ext = '.DXF') or
           (ext = '.E00') or (ext = '.DWG');
end;

procedure TConversor.tb_ImportarClick(Sender: TObject);
var
   aShellFolder: TShellFolder;
   aListItem : TListitem;
begin
  aListItem:= slv_Files.Selected;

  while aListItem <> nil do
  begin
    aShellFolder:= slv_Files.Folders[aListItem.Index];
    insertFile(aShellFolder.PathName);
    aListItem:= slv_Files.GetNextItem(aListItem,sdAll,[isSelected]);
  end;

end;

procedure TConversor.insertFile(aPath: string);
var ext :string;

  procedure createList(aNode :TTreeNode);
  var aFileList :TStringList;
  begin
    aFileList:= TStringList.Create;
    aFileList.Sorted:= true;
    aFileList.Duplicates:= dupIgnore;
    aNode.Data:= aFileList;
  end;

begin
  ext:= upperCase(ExtractFileExt(aPath));
  if ext = '.MIF' then
  begin
    if nodeMIF.Data = nil then
      createList(nodeMIF);
    TStringList(nodeMIF.Data).Add(aPath);
  end else
```

```
  if ext = '.SHP' then
  begin
    if nodeSHP.Data = nil then
      createList(nodeSHP);
    TStringList(nodeSHP.Data).Add(aPath);
  end else

  if ext = '.SDL' then
  begin
    if nodeSDL.Data = nil then
      createList(nodeSDL);
    TStringList(nodeSDL.Data).Add(aPath);
  end;
end;

procedure TConversor.PageControl1Change(Sender: TObject);
var j      :Integer;
    fileName :string;
    tempSL   :TStringList;
    tempNode :TTreeNode;

  procedure insertFileFields(aFormatNode :TTreeNode; aPath :string; parentNode :TTreeNode);
  var fieldList :TStringList;
      i         :Integer;
  begin
    if aFormatNode = nodeMIF then
      fieldList:= MIDMIFParser.getColumnNames(MIDMIFParser.getHeader(aPath),true)
    else
    if aFormatNode = nodeSDL then
      fieldList:= SDLParser.getColumnNames
    else
    if aFormatNode = nodeSHP then
    begin
      DBFParser.Open( ChangeFileExt(aPath,'.dbf') );
      fieldList:= SHPParser.getColumnNames(true);
    end;

    for i:= 0 to fieldList.Count -1 do
      tv_Formats.Items.AddChild(parentNode,fieldList[i]);
  end;

begin
  if PageControl1.ActivePageIndex = 1 then
  begin
//PINTER Adiciono os arquivos na árvore do formato..
  tempNode:= tv_Formats.Items[0];
    while tempNode <> nil do                //lista de formatos
    begin
      if tempNode.Data <> nil then
      begin
        tempSL:= TStringList(tempNode.Data);
        for j:= 0 to tempSL.Count -1 do      //lista dos arquivos do formato
          if tempSL.Objects[j] = nil then
          begin
            fileName:= tempSL[j];
            tempSL.Objects[j]:= tv_Formats.Items.AddChild(tempNode,extractFileName(fileName));
            insertFileFields(tempNode,fileName,TTreeNode(tempSL.Objects[j]));
          end;
      end;
      tempNode:= tempNode.getNextSibling;
    end;
  end;
end;

procedure TConversor.FormCreate(Sender: TObject);
begin
```

```
  nodeMIF:= tv_Formats.Items[4];
  nodeSDL:= tv_Formats.Items[6];
  nodeSHP:= tv_Formats.Items[7];

  with MIDMIFParser do begin
    NodeClass:= TCustomNode;
    LineClass:= TCustomLine;
    AreaClass:= TCustomArea;
    exportIgnoredObjects:= false;
  end;
  with SDLParser do begin
    NodeClass:= TCustomNode;
    LineClass:= TCustomLine;
    AreaClass:= TCustomArea;
    exportIgnoredObjects:= false;
  end;
  with SHPParser do begin
    NodeClass:= TCustomNode;
    LineClass:= TCustomLine;
    AreaClass:= TCustomArea;
    exportIgnoredObjects:= false;
  end;
  setScaleUp;
  layerList:= TStringList.Create;
end;

procedure TConversor.cb_SaveUnsuportedClick(Sender: TObject);
begin
  label3.Enabled:= cb_SaveUnsuported.Checked;
  DirectoryEdit1.Enabled:= cb_SaveUnsuported.Checked;
  MIDMIFParser.exportIgnoredObjects:= cb_SaveUnsuported.Checked;
  SHPParser.exportIgnoredObjects:= cb_SaveUnsuported.Checked;
  SDLParser.exportIgnoredObjects:= cb_SaveUnsuported.Checked;
  if cb_SaveUnsuported.Checked then DirectoryEdit1.Text:= fe_FileDestiny.Text;
end;

procedure TConversor.Button1Click(Sender: TObject);
var i:integer;
begin
  try
    loadFromFormatFiles;
    storeIntoFormatFiles;
  except
    MessageDlg('Erro ao converter os Dados!!'+#10+'Confira se os parâmetros setados estão corretos.',
                    mtError, [mbAbort], 0);
    exit;
  end;
  Button2Click(self);
  PageControl1.ActivePageIndex:= 0;
  clearFormatTree;
  MessageDlg('Conversão realizada com sucesso!!',mtInformation,[mbOK], 0);
end;

procedure TConversor.storeIntoFormatFiles;
var i          :integer;
    listOfData  :TStringList;
begin
  setScaleDown;
  listOfData:= TStringList.Create;
  if rb_Midmif.Checked then           //Destino MIF
    for i:= 0 to layerList.Count-1 do
    begin
      if ExtractFileExt(layerList[i]) = '.mif' then  //Origem MIF
        MIDMIFParser.povoateFieldsDefs(
MIDMIFParser.getColumnNames(MIDMIFParser.getHeader(layerList[i]),true),ffMidMif )
      else
        if ExtractFileExt(layerList[i]) = '.shp' then   //Origem SHP
```

```
      begin
        DBFParser.Open( ChangeFileExt(layerList[i],'.dbf') );
        MIDMIFParser.povoateFieldsDefs( SHPParser.getColumnNames(true) ,ffSHP);
        DBFParser.Close;
      end else                                  //Origem SDL
      begin
        MIDMIFParser.XScaleFactor:= 0.01;
        MIDMIFParser.YScaleFactor:= 0.01;
        MIDMIFParser.povoateFieldsDefs( SDLParser.getColumnNames ,ffSDL);
      end;
    MIDMIFParser.storeMIF(fe_FileDestiny.Text + ExtractFileName( ChangeFileExt(layerList[i],'.mif') ),
                    TCustomLayer(layerList.Objects[i]));
    listOfData.LoadFromFile(ExtractFilePath(layerList[i]) + '_accepted_' +
                     ExtractFileName( ChangeFileExt(layerList[i],'.csv') ));
    listOfData.SaveToFile(fe_FileDestiny.Text + ExtractFileName( ChangeFileExt(layerList[i],'.mid') ));
   end
  else

   if rb_Shpdbf.Checked then       //Destino SHP
    for i:= 0 to layerList.Count-1 do
    begin
      if ExtractFileExt(layerList[i]) = '.mif' then       //Origem MIF
       SHPParser.StoreDBFFromList(layerList[i],fe_FileDestiny.Text,
         MIDMIFParser.getColumnNames(MIDMIFParser.getHeader(layerList[i])),ffMidMif)
      else
       if ExtractFileExt(layerList[i]) = '.shp' then     //Origem SHP
       begin
         DBFParser.Open( ChangeFileExt(layerList[i],'.dbf') );
         SHPParser.StoreDBFFromList(layerList[i],fe_FileDestiny.Text,SHPParser.getColumnNames(true),ffSHP);
         DBFParser.Close
       end
       else                                  //Origem SDL
       begin
         SHPParser.XScaleFactor:= 0.01;
         SHPParser.YScaleFactor:= 0.01;
         SHPParser.StoreDBFFromList(layerList[i],fe_FileDestiny.Text,SDLParser.getColumnNames,ffSDL);
       end;
      SHPParser.storeSHP(fe_FileDestiny.Text + ExtractFileName( ChangeFileExt(layerList[i],'.shp') ),
                    TCustomLayer(layerList.Objects[i]));
    end
   else
    for i:= 0 to layerList.Count-1 do
     SDLParser.store(fe_FileDestiny.Text + ExtractFileName( ChangeFileExt(layerList[i],'.sdl') ),
                    TCustomLayer(layerList.Objects[i]));
  listOfData.Free;
  setScaleUp;
end;

procedure TConversor.loadFromFormatFiles;
var i :Integer;
   tempSL :TStringList;
begin
//Leitura dos dados em MIFs
  if (tv_Formats.Checked[nodeMIF.AbsoluteIndex]) and (nodeMIF.Data <> nil) then
  begin
    tempSL:= TStringList(nodeMIF.Data);
    for i:= 0 to tempSL.Count -1 do
    begin
      if cb_SaveUnsuported.Checked then
        MIDMIFParser.exportIgnoredFileName:=
          DirectoryEdit1.Text + ExtractFileName( ChangeFileExt(tempSL[i],'.lix') );
     layerList.AddObject(lowerCase(tempSL[i]),MIDMIFParser.loadLayer(DestinyObjKind,tempSL[i]));
    end;
  end else
//Leitura dos dados em SHPs
  if (tv_Formats.Checked[nodeSHP.AbsoluteIndex]) and (nodeSHP.Data <> nil) then
  begin
```

```
    tempSL:= TStringList(nodeSHP.Data);
    for i:= 0 to tempSL.Count -1 do
    begin
      if cb_SaveUnsuported.Checked then
        SHPParser.exportIgnoredFileName:=
          DirectoryEdit1.Text + ExtractFileName( ChangeFileExt(tempSL[i],'.lix') );
      layerList.AddObject(lowerCase(tempSL[i]),SHPParser.loadLayer(DestinyObjKind,tempSL[i]));
    end;
  end else
//Leitura dos dados em SDLs
  if (tv_Formats.Checked[nodeSDL.AbsoluteIndex]) and (nodeSDL.Data <> nil) then
  begin
    tempSL:= TStringList(nodeSDL.Data);
    for i:= 0 to tempSL.Count -1 do
    begin
      if cb_SaveUnsuported.Checked then
        SDLParser.exportIgnoredFileName:=
          DirectoryEdit1.Text + ExtractFileName( ChangeFileExt(tempSL[i],'.lix') );
      layerList.AddObject(lowerCase(tempSL[i]),SDLParser.loadLayer(DestinyObjKind,tempSL[i]));
    end;
  end;
  for i:= 0 to layerList.Count -1 do
    TCustomLayer(layerList.Objects[i]).BuildBoundRect;
end;

procedure TConversor.ObjKindClick;
begin
  if RadioButton1.Checked then DestinyObjKind:= okPoint
  else
    if RadioButton2.Checked then DestinyObjKind:= okLine
    else DestinyObjKind:= okArea;
end;

procedure TConversor.RadioButton1Click(Sender: TObject);
begin
  ObjKindClick;
end;

procedure TConversor.FormDestroy(Sender: TObject);
var i :integer;
begin
  for i:= 0 to layerList.Count -1 do TCustomLayer(layerList.Objects[i]).destroy;
  layerList.Free;
end;

procedure TConversor.fe_FileDestinyExit(Sender: TObject);
var i :integer;
begin
  with TDirectoryEdit(sender) do
  begin
    i:= Length(Text);
    if Text[i] <> '\' then Text:= Text + '\';
  end;
end;

procedure TConversor.Button2Click(Sender: TObject);
begin
  rb_Midmif.Checked:= false;
  rb_Shpdbf.Checked:= false;
  rb_Sdl.Checked:= false;
  radioButton1.Checked:= false;
  radioButton2.Checked:= false;
  radioButton3.Checked:= false;
  fe_FileDestiny.Text:= '';
  cb_SaveUnsuportedClick(cb_SaveUnsuported);
end;
```

```
procedure TConversor.setScaleDown;
begin
  with MIDMIFParser do begin
    XScaleFactor:= flScaleDown;
    YScaleFactor:= flScaleDown;
  end;
  with SDLParser do begin
    XScaleFactor:= 0.01;
    YScaleFactor:= 0.01;
  end;
  with SHPParser do begin
    XScaleFactor:= flScaleDown;
    YScaleFactor:= flScaleDown;
  end;
end;

procedure TConversor.setScaleUP;
begin
  with MIDMIFParser do begin
    XScaleFactor:= flScaleUp;
    YScaleFactor:= flScaleUp;
  end;
  with SDLParser do begin
    XScaleFactor:= 100;
    YScaleFactor:= 100;
  end;
  with SHPParser do begin
    XScaleFactor:= flScaleUp;
    YScaleFactor:= flScaleUp;
  end;
end;

procedure TConversor.clearFormatTree;
var i :integer;
begin
  nodeMIF.DeleteChildren;
  tv_Formats.Checked[nodeMIF.AbsoluteIndex]:= false;

  nodeSHP.DeleteChildren;
  tv_Formats.Checked[nodeSHP.AbsoluteIndex]:= false;

  nodeSDL.DeleteChildren;
  tv_Formats.Checked[nodeSDL.AbsoluteIndex]:= false;
end;

end.
```

```
unit ooGISParseFileFormat;

interface
uses classes,contnrs,ooGIS,ooGISCustomObjects,DB,DBClient;

type

TExtractMethod = function(i : integer) : TCustomLayerItem of object;

TCustomGISFormatParser = class
private
protected
  FNodeClass : TNodeClass;
  FLineClass : TLineClass;
  FAreaClass : TAreaClass;

  FExportIgnored : Boolean;
  FExportIgnoredFilename : string;

  FBinFile : TFileStream; //shp file format
  FTextFile : TextFile; //sdl, mif file format
  FFields : array of TField;
  FDataSet : TClientDataSet;

  FNodes : TObjectList;
  FLines : TObjectList;
  FAreas : TObjectList;

  FCoordinateSystem : TCoordinateSystem;
  FHemisphere : THemisphere;
  FXScaleFactor : double;
  FYScaleFactor : double;
  FZone : integer;

  function    getNodes : integer;
  function    getLines : integer;
  function    getAreas : integer;

  function    getNode(i : Integer) : TCustomNode;
  function    getLine(i : Integer) : TCustomLine;
  function    getArea(i : Integer) : TCustomArea;

  function    readCoordinate(newLine : boolean) : TCoordinate; virtual;
  procedure   writeCoordinate(const coord : TCoordinate; newLine : Boolean); virtual;

  function  CoordinateToStr(Coord : TCoordinate) : string;
public
  constructor create; virtual;
  destructor  destroy; override;

  procedure   setField(i :integer; aField :TField);
  procedure   clear;

  procedure   load(FileName : string); virtual; abstract; //load data from FORMAT file(s)
  procedure   store(FileName : string; Layer : TCustomLayer); virtual; abstract; //store data to FORMAT file(s)

  function    loadLayer(Mode : TObjectKind; filename : String) : TCustomLayer; virtual; abstract;

  procedure   setDataSource(dataSet : TClientDataset; usedFields : TByteSet); virtual;

  function    extractNode(i : integer) : TCustomLayerItem; virtual;
  function    extractLine(i : integer) : TCustomLayerItem; virtual;
  function    extractArea(i : integer) : TCustomLayerItem; virtual;

  property    NodeClass : TNodeClass read FNodeClass write FNodeClass;
  property    LineClass : TLineClass read FLineClass write FLineClass;
  property    AreaClass : TAreaClass read FAreaClass write FAreaClass;
```

```
  property    XScaleFactor : double write FXScaleFactor;
  property    YScaleFactor : double write FYScaleFactor;

  property    Areas : integer read getAreas;
  property    Area[i : integer] : TCustomArea read getArea;
  property    Lines : integer read getLines;
  property    Line[i : integer] : TCustomLine read getLine;
  property    Nodes : integer read getNodes;
  property    Node[i : integer] : TCustomNode read getNode;

  property    exportIgnoredObjects : Boolean write FExportIgnored;
  property    exportIgnoredFileName : string write FExportIgnoredFileName;
end;


implementation

// 8<---------------------------------------------------------------------------
uses SysUtils;

constructor TCustomGISFormatParser.create;
begin
  FNodes:= TObjectList.Create;
  FLines:= TObjectList.Create;
  FAreas:= TObjectList.Create;
end;

destructor  TCustomGISFormatParser.destroy;
begin
  FFields:= nil;

  FNodes.free;
  FLines.free;
  FAreas.free;

  inherited;
end;

function TCustomGISFormatParser.getNodes;
begin
  result:= FNodes.count;
end;

function TCustomGISFormatParser.getLines;
begin
  result:= FLines.count;
end;

function TCustomGISFormatParser.getAreas;
begin
  result:= FAreas.count;
end;

function TCustomGISFormatParser.getNode;
begin
  if i >= FNodes.Count
    then result:= nil
    else result:= TCustomNode(FNodes[i]);
end;

function TCustomGISFormatParser.getLine;
begin
  if i >= FLines.Count
    then result:= nil
    else result:= TCustomLine(FLines[i]);
end;
```

```
function TCustomGISFormatParser.getArea;
begin
  if i >= FAreas.Count
    then result:= nil
    else result:= TCustomArea(FAreas[i]);
end;

function TCustomGISFormatParser.ReadCoordinate;
var
   a,b : double;
begin
  read(FTextFile,a,b);
  result.Lon:= TRUNC(a * FXScaleFactor);
  result.Lat:= TRUNC(b * FYScaleFactor);
  if newLine then readLN(FTextFile);
end;

procedure  TCustomGISFormatParser.writeCoordinate;
begin
  writeLN(FTextFile,CoordinateToStr(coord));
end;

procedure TCustomGISFormatParser.clear;
begin
  FNodes.Clear;
  FLines.Clear;
  FAreas.Clear;
end;

procedure TCustomGISFormatParser.SetDataSource;
var
   i,k : integer;
begin
  FFields:= nil;
  FDataset:= dataset;
  if FDataSet = nil then exit;

  with FDataset do begin
    setLength(FFields,Fields.Count);
    k:= 0;
    for i:= 0 to Fields.Count-1 do
      if (usedFields = []) or (i in usedFields) then begin
        FFields[k]:= Fields[i];
        inc(k);
      end;
    if k < Fields.Count then setLength(FFields,k);
  end;
end;

function TCustomGISFormatParser.ExtractNode;
begin
  result:= node[i];
  FNodes.extract(result);
end;

function TCustomGISFormatParser.ExtractLine;
begin
  result:= line[i];
  FLines.extract(result);
end;

function TCustomGISFormatParser.ExtractArea;
begin
  result:= area[i];
  FAreas.extract(result);
end;
```

```
procedure TCustomGISFormatParser.setField(i: integer; aField: TField);
begin
  FFields[i]:= aField;
end;

function TCustomGISFormatParser.CoordinateToStr;
begin
  result:= format('%.8f %.8f',[coord.lon * FXScaleFactor,coord.lat * FYScaleFactor]);
end;

end.
```

```
unit ooGISCustomObjects;

interface
uses
  Classes,
  Windows,
  Graphics,
  ooGIS,
  ooGisLayers;


// 8<-----------------------------------------------------------------------------
// Implementações da classe MIFObjects, que adiciona capacidades de tratamento
// de arquivos MIF
// 8<-----------------------------------------------------------------------------

type

TArrayOfInteger = array [1..128] of Integer;
PArrayOfInteger = ^TArrayOfInteger;


TGISObject = class(TCustomLayerItem)
protected
  FLabel : string;
  FCoordinates : PArrayOfCoordinates;
  FCoordCount : Integer;

  function     GetSize : String; virtual; {abstract;}
public
  class function Name : String; virtual; abstract;

  constructor    create(Version : integer); virtual; abstract;
  destructor     destroy; override;

  property       size : String read getSize;
end;

TPoint = Class(TGISObject)
protected
  function     GetCoordinate : TCoordinate;
  procedure    SetCoordinate(c : TCoordinate);
public
  class function Name : String; override;

  constructor  Create(Version : integer); override;
  constructor  CreateFromCoord(const a : TCoordinate); virtual;

  procedure    AssignCoordinates(origin : TCustomLayerItem); override;

  procedure    WriteToList(const lines : TStrings); override;

  property     Coordinate : TCoordinate read getCoordinate write setCoordinate;
end;

TLine = Class(TGISObject)
protected
 function     GetSize : string; override;
 function     GetCoordinate(i : Integer) : TCoordinate;
public
 class function Name : String; override;

 constructor  Create(Version : integer); override;
 constructor  CreateFromCoords(const a,b : TCoordinate); virtual;

 procedure    AssignCoordinates(origin : TCustomLayerItem); override;
 procedure    WriteToList(const lines : TStrings); override;
```

```
 property     Coordinate[i : Integer] : TCoordinate read getCoordinate; default;
end;

//----Classe pai de TPLine e TRegion-
//    TSectionObject: Tem praticamente a estrutura da antiga TPline

TSectionObjectClass = class of TSectionObject;

TSectionObject = class(TGISObject)
protected
  FSections      : Integer;
  FData          : TStringList;
  FDeepest       : Integer;
  FBounds        : TRect;
  FSectionBounds : TList;

  function     getSize : String; override;
  function     getDeep(i : Integer) : Integer; virtual;

  function     getCoordinate(i,j : integer) : TCoordinate; virtual;
  procedure    setCoordinate(i,j : integer; c : TCoordinate); virtual;

  function     getSectionData(i : integer) : Pointer; virtual;
  function     getSectionBound(i : Integer) : TRect;  virtual;

  procedure    writeListObjectHeader(const lines : TStrings); virtual; abstract;

  function     readRectangle(var arq : TextFile; newLine : boolean) : TRect; virtual;
  procedure    writeRectangle(var arq : TextFile; r : TRect; newLine : boolean); virtual;

  procedure    killBounds;
public
  constructor  create(SectionCount : integer); reintroduce; virtual;
  destructor   destroy; override;

  procedure    AssignCoordinates(origin : TCustomLayerItem); override;

  procedure    AddSection(sectionSize : Integer);
  procedure    AddSectionRectangle(rectangle : TRect);
  procedure    BuildSectionRectangle(sectionNumber : integer; mainRectangle : boolean);

  procedure    WriteToList(const lines : TStrings); override;
// 0           - significa o bounding do objeto todo
// 1..FSections - se entrar um valor neste intervalo constroi o bound da seção
  function     BuildBounds(section : Integer) : TRect; virtual;
//estas funções executam duas importantes metamorfoses em um PLINE, como em toda
//boa metamorfose a criatura original desaparece, isto é, estas funções destroem
//o PLine original
  function     MetamorfoseTo(resultClass : TSectionObjectClass) : TSectionObject; virtual; abstract;

  property     BoundRect : TRect read FBounds write FBounds;
// Tanto (i), quanto (j) tem como o menor valor a unidade, isto é, 1 (um).
// O (i) representa a i-ésima coordenada da j-ésima seção
  property     Coordinate[i,j : Integer] : TCoordinate read getCoordinate write SetCoordinate; Default;
//armazena a maior seção e é utilizada para acelerar as alocações de memória na
//hora de desenhar, sendo determinada no momento da leitura do arquivo
  property     Deepest : Integer read FDeepest;
  property     Sections : Integer read FSections;
//serve para pegar o vetor todo de uma vez
  property     SectionData[i : integer] : Pointer read getSectionData;
//serve para pegar o número de items em cada section
  property     SectionDeep[i : Integer] : Integer read GetDeep;
  property     SectionBounds[i : integer] : TRect read getSectionBound;
end;

//-------------------------------------
```

```
  TRegion = class(TSectionObject)
  protected
    FCenter : TCoordinate;

    procedure writeListObjectHeader(const lines : TStrings); override;
  public
    class function name : String; override;

    constructor  create(SectionCount : integer); override;
//este constructor é utilizado para criar uma região utilizando coordenadas de
//armazenadas em outro objeto, notar que as coordenadas não são duplicadas apenas
//os ponteiros para os vetores é que serão, cuidado para não liberar duas vezes
//as coordenadas
    constructor  CreateFromCoords(const coords : TStringList);

    procedure    AssignCoordinates(origin : TCustomLayerItem); override;

    procedure    RotateCoordinates(section, newFirst: integer);

    property     Center : TCoordinate read FCenter write FCenter;
  end;

//----------
  TRectangle = class(TSectionObject)
  protected
   procedure writeListObjectHeader(const lines : TStrings); override;

  public
    class Function Name :String; override;
    constructor create;

  end;


//----------

const

  ltSingleLine     = 0;
  ltSinglePolyline = 1;
  ltMultiPolyline  = 2;

type

  TPLine  = class(TSectionObject)
  protected
     procedure writeListObjectHeader(const lines : TStrings); override;

  public
    class function Name : String; override;

    constructor  create(SectionCount : integer); override;

    procedure    connect(point : integer; coordinate : TCoordinate);

    function     metamorfoseToLINE : TLine;
    function     metamorfoseToREGION : TRegion;
  end;

//----------

  TText = class(TGISObject)
  protected
     FText : String;

     function  getSize : String; override;
     function  getCoordinate(i : integer) : TCoordinate;
```

```
  public
    class function Name : String; override;

    constructor  create(Version : integer); override;

    procedure    WriteToList(const lines : TStrings); override;

    property     Caption : String read FText write FText;
    property     Coordinate[i : Integer] : TCoordinate read getCoordinate; default;
  end;

// 8<-------------------------------------------------------------------------
// The real stuff, estas serão as classe que realmente serão utilizadas dentro
// do programa os ícones das camadas de pontos no mapa
// 8<-------------------------------------------------------------------------

TCustomNode = class(TPoint)
private
  FIdCode: string;
  FIdLabel: string;
protected
  function    GetIDCode: string; override;
  procedure   SetIDCode(cod: string); override;

  function    GetIDLabel: string; override;
  procedure   SetIDLabel(cod: string); override;
public
  constructor CreateWithLayer(ALayer : TCustomLayer); override;
end;

// .....................................................................

TCustomLine = class(TPline)
private
  FIdCode: LongWord;
  FIdLabel: string;
protected
  procedure setCoordinate(i,j : Integer; c : TCoordinate);
  function  GetIDCode: string; override;
  procedure SetIDCode(cod: string); override;

  function    GetIDLabel: string; override;
  procedure   SetIDLabel(cod: string); override;
public
  constructor create(SectionCount : integer); override;
  constructor createWithLayer(ALayer : TCustomLayer); override;
  procedure   SetSizes(Deeps : array of Integer); virtual;
// Tanto (i), quanto (j) tem como o menor valor a unidade, isto é, 1 (um).
// O (i) representa a i-ésima coordenada da j-ésima seção
  property    Coordinate[i,j : Integer] : TCoordinate read getCoordinate write setCoordinate; Default;
end;

// .....................................................................

TCustomArea = class(TRegion)
private
  FIDCode: string;
  FIDLabel: string;
protected
  function    GetIDCode: string; override;
  procedure   SetIDCode(cod: string); override;

  function    GetIDLabel: string; override;
  procedure   SetIDLabel(cod: string); override;
public
  constructor CreateWithLayer(ALayer : TCustomLayer); override;
  procedure   WriteBounds(var arq : TextFile; s : Integer);
```

```
end;

// 8<----------------------------------------------------------------------------

TNodeClass = class of TCustomNode;

TLineClass = class of TCustomLine;

TAreaClass = class of TCustomArea;

implementation
uses SysUtils, StrUtils, ooStamper, dialogs;

//-------------------------------------------------------------------------------------------------
// CLASSE MIF OBJECT, abstrata, implementa apenas o destructor e a função para ler um par lat/lon
//
//-------------------------------------------------------------------------------------------------

destructor TGISObject.destroy;
begin
  if FCoordCount > 0 then FreeMem(FCoordinates,fCoordCount * SizeOF(TCoordinate));
  inherited
end;

function   TGISObject.getSize;
begin
  Result:= '?!?';
end;


//-------------------------------------------------------------------------------------------------
// CLASSE POINT
//
//-------------------------------------------------------------------------------------------------

class function TPoint.Name;
begin
  result:= tagPOINT;
end;

constructor  TPoint.create;
begin
  FCoordCount:= 1;
  GetMem(FCoordinates,SizeOf(TCoordinate));
end;

constructor TPoint.createFromCoord;
begin
  FCoordCount:= 1;
  GetMem(FCoordinates,SizeOf(TCoordinate));
  FCoordinates^[1]:= a;
end;

function    TPoint.GetCoordinate;
begin
  result:= FCoordinates^[1];
end;

procedure   TPoint.SetCoordinate;
begin
  FCoordinates^[1]:= c;
end;

procedure    TPoint.AssignCoordinates(origin : TCustomLayerItem);
var
   point : TPoint absolute origin;
begin
```

```
  FCoordinates^[1]:= point.Coordinate;
end;

procedure    TPoint.WriteToList(const lines : TStrings);
var
   s : String;
begin
  s:= format('%s %s',[name,CoordinateToStr(FCoordinates^[1])]);
  lines.append(s);
  // se tiver...
end;

//-----------------------------------------------------------------------------------------------------
// CLASSE LINE
//
//-----------------------------------------------------------------------------------------------------

class function TLine.Name;
begin
  result:= tagLINE;
end;

constructor  TLine.create;
{}
  procedure extract(Var coord : TCoordinate);
  var
     p : Integer;
     v : String;
  begin
(*    Tag:= trim(tag);
    p:= Pos(' ',tag);
    v:= system.copy(tag,1,p-1);
    coord.lat:= TRUNC(StrToFloat(v) * flScaleUP);
    system.delete(tag,1,p);
    Tag:= trim(tag);
    p:= Pos(' ',tag);
    if p = 0 then p:= length(tag) + 1;
    v:= system.copy(tag,1,p-1);
    coord.lon:= TRUNC(StrToFloat(v) * flScaleUP);
    system.delete(tag,1,p);  *)
  end;
{}
begin
  FCoordCount:= 2;
  GetMem(FCoordinates,SizeOf(TCoordinate)*2);
(*  system.delete(tag,1,Pos(' ',tag));
  extract(FCoordinates^[1]);
  extract(FCoordinates^[2]);*)
end;

constructor  TLine.CreateFromCoords;
begin
  FCoordCount:= 2;
  GetMem(FCoordinates,SizeOf(TCoordinate)*2);
  FCoordinates^[1]:= a;
  FCoordinates^[2]:= b;
end;

function     TLine.getSize;
begin
  Result:= Format('(%s %s)',[CoordinateToStr(FCoordinates^[1]),CoordinateToStr(FCoordinates^[2])]);
end;

function     TLine.GetCoordinate;
begin
  result:= FCoordinates^[i];
end;
```

```
procedure    TLine.AssignCoordinates(origin : TCustomLayerItem);
var
   line : TLine absolute origin;
begin
  FCoordinates^[1]:= line.coordinate[1];
  FCoordinates^[2]:= line.coordinate[2];
end;

procedure    TLine.WriteToList;
var
   s : String;
begin
  s:= format('%s %s %s',[name,CoordinateToStr(FCoordinates^[1]),CoordinateToStr(FCoordinates^[2])]);
  lines.append(s);
  //se tiver PEN incluir
end;

//---------------------------------------------------------------------------------------------------
// CLASSE RECTANGLE
//
//---------------------------------------------------------------------------------------------------


//---------------------------------------------------------------------------------------------------
// CLASSE ELIPPSE
//
//---------------------------------------------------------------------------------------------------

//---------------------------------------------------------------------------------------------------
// CLASSE ROUND RECTANGLE
//
//---------------------------------------------------------------------------------------------------

//---------------------------------------------------------------------------------------------------
// CLASSE Section Object
//
//---------------------------------------------------------------------------------------------------

function    TSectionObject.BuildBounds;
var
   i,j : integer;
   r   : TRect;
   cs  : PArrayOfCoordinates;
{}
  function BuildBound : TRect;
  var
     i : integer;
  begin
    with result do begin
      {}
      left  :=  1000000000;
      top   := -1000000000;
      right := -1000000000;
      bottom:=  1000000000;
      {}
      j:= SectionDeep[section];
      {}
      for i:= 1 to j do with cs^[i] do begin
//DETT 15/07/1999
// Pesquisa das coordenadas do bound rect
//DETT 21/07/1999 ATUALIZAÇÃO
// Os if-then-else originais foram desaninhados
        if lat < bottom then bottom:= lat;
        if lat > top then top:= lat;
        if lon > right then right:= lon;
        if lon < left then left:= lon;
```

```
      end;
    end;
  end;
{}
begin
  if section = 0
    then begin
      result:= getSectionBound(1);
      if FSections > 1 then for i:= 2 to FSections do begin
        r:= getSectionBound(i);
        with Result do begin
          if r.Left < Left then Left:= r.Left;
          if r.Right > Right then Right:= r.Right;
          if r.Top > Top then Top:= r.Top;
          if r.Bottom < Bottom then Bottom:= r.Bottom;
        end;
      end;
    end
    else begin
//DETT 2003/04/02
//Adicionei a atribuição de FCoordinates;
      cs:= SectionData[section];
      result:= buildBound;
    end;
end;

procedure   TSectionObject.addSection;
begin
  GetMem(FCoordinates,sizeOf(TCoordinate)*SectionSize); //aloco memória para as coordenadas
  FData.addObject(intToStr(sectionSize),Pointer(FCoordinates));
  FSections:= FData.Count;
  {}
  if SectionSize > FDeepest then FDeepest:= SectionSize;
end;

procedure   TSectionObject.AddSectionRectangle(rectangle : TRect);
var
   r : ^TRect;
begin
  new(r);
  r^:= rectangle;
  FSectionBounds.add(r);
end;

procedure   TSectionObject.BuildSectionRectangle;
var
   r : ^TRect;
begin
  if mainRectangle
    then FBounds:= BuildBounds(sectionNumber)
    else begin
      new(r);
      r^:= BuildBounds(sectionNumber);
      FSectionBounds.add(r);
    end;
end;

// 8<-------------------------------------------------------------------------

procedure   KillDataItems(list : TStringList);
var
   i,j : integer;
   p : Pointer;
begin
  with list do if count > 0 then begin
    for i:= 0 to count - 1 do begin
      j:= strToInt(list[i]);
```

```
      p:= objects[i];
      freeMem(p,sizeOf(TCoordinate)*j);
    end;
    clear;
  end;
end;

procedure TSectionObject.KillBounds;
var
   i : integer;
   r : ^TRect;
begin
  with FSectionBounds do begin
    if count > 0 then for i:= 0 to count - 1 do begin
      r:= items[i];
      Dispose(r);
    end;
    clear;
  end;
end;

constructor  TSectionObject.create;
begin
  FSections:= SectionCount;
  FData:= TStringList.Create;
  FSectionBounds:= TList.Create;
end;

destructor  TSectionObject.destroy;
begin
  KillDataItems(FData);
  FData.free;
  {}
  KillBounds;
  FSectionBounds.Free;
  {}
  inherited;
end;

function  TSectionObject.GetSize;
var
   i,j,k : integer;
begin
  k:= 0;
  for i:= 0 to FData.count-1 do begin
    j:= strToInt(FData[i]);
    inc(k,j);
  end;
  result:= format('(%3.3d/%3.3d)',[FSections,k]);
end;

function    TSectionObject.getDeep;
begin
  result:= 0; //default
  if (i > 0) and (i <= FSections) then result:= strToInt(FData[i-1]);
end;

procedure   TSectionObject.setCoordinate;
begin
  try
    FCoordinates:= pointer(FData.objects[j-1]);
    FCoordinates^[i]:= c;
  except on
    Exception do FCoordinates^[i]:= CoordZero;
  end;
end;
```

```
function    TSectionObject.GetCoordinate;
begin
  try
    FCoordinates:= pointer(FData.objects[j-1]);
    result:= FCoordinates^[i];
  except on
    Exception do result:= CoordZero;
  end;
end;

function  TSectionObject.getSectionData;
begin
  if i in [1..FSections]
    then result:= FData.objects[i-1]
    else result:= nil;
end;

function  TSectionObject.GetSectionBound;
begin
  if (i in [1..FSections]) and (FSections > 1)
    then Result:= TRect(FSectionBounds[i-1]^)
    else Result:= FBounds;
end;

function  TSectionObject.readRectangle;
var
   a,b,c,d : Double;
begin
  read(arq,a,b,c,d);
  with Result do begin
    Left:= trunc(a * flScaleUP);
    Top:= trunc(b * flScaleUP);
    Right:= trunc(c * flScaleUP);
    Bottom:= trunc(d  * flScaleUP);
  end;
  if newLine then readLN(arq);
end;

procedure TSectionObject.writeRectangle;
begin
  with r do begin
    write(arq,(Left * flScaleDOWN):0:8,' ');
    write(arq,(Top * flScaleDOWN):0:8,' ');
    write(arq,(Right * flScaleDOWN):0:8,' ');
    write(arq,(Bottom * flScaleDOWN):0:8);
  end;
  if newLine then writeLN(arq);
end;

procedure TSectionObject.WriteToList;
var
   i,j,k : Integer;
   s : string;
begin
  writeListObjectHeader(lines);
  for k:= 0 to FSections-1 do begin
    lines.append(' '+FData[k]); //escrevo o número de pontos na seção atual
    i:= strToInt(FData[k]);
    FCoordinates:= Pointer(FData.objects[k]);
    for j:= 1 to i do begin
      s:= ' ' + coordinateToStr(FCoordinates^[j]);
      lines.append(s);
    end;
  end;
  //se tiver STYLE deverá escrever também  etc
end;
```

```
procedure   TSectionObject.AssignCoordinates(origin : TCustomLayerItem);
var
   i,j : integer;
   pline : TSectionObject absolute origin;
   p : Pointer;
   r : ^TRect;
begin
  KillDataItems(FData);
  {}
  with FSectionBounds do for i:= 0 to count-1 do begin
    r:= items[i];
    Dispose(r);
  end;
  FSectionBounds.Clear;
  {}
  FSections:= pline.sections;
  FBounds:= pline.FBounds;
  FDeepest:= pline.deepest;
  for i:= 1 to pline.sections do begin
    j:= pLine.Sectiondeep[i];
    GetMem(FCoordinates,sizeOf(TCoordinate)*j); //aloco memória para as coordenadas
    FData.addObject(intToStr(j),Pointer(FCoordinates));
    p:= pLine.FData.objects[i-1]; //isto só é possível porque estou na própria unit que define PLINE
    move(p^,FCoordinates^,j * sizeOf(TCoordinate));
    {}
    if pline.sections > 1 then begin
      new(r);
      r^:= TRect(pline.FSectionBounds[i-1]^); //isto aqui também
      FSectionBounds.add(r);
    end;
    {}
  end;
end;

//-----------------------------------------------------------------------------------------------------
// CLASSE POLY LINE
//
//-----------------------------------------------------------------------------------------------------

class function TPLine.Name;
begin
  result:= tagPOLYLINE;
end;

constructor  TPLine.create;
begin
  inherited;
end;

procedure    TPLine.writeListObjectHeader;
var
    s : string;
begin
  s:= name;
  if FSections > 1 then s:= s + ' ' + tagMultiple + ' ' + intToStr(FSections);
  lines.append(s);
end;

procedure    TPline.Connect;
{var
   i : integer;}
begin
  if point = 0 //extremo inicial
    then begin
      FCoordinates:= Pointer(FData.objects[0]);
      FCoordinates^[1]:= coordinate;
```

```
    end
    else begin
      FCoordinates:= Pointer(FData.objects[FSections-1]);
      FCoordinates^[StrToInt(FData[FSections-1])]:= coordinate;
    end;
end;

function     TPLine.MetamorfoseToLINE;
begin
  Result:= TLine.CreateFromCoords(Coordinate[1,1],Coordinate[SectionDeep[1],1]);
  Free;
end;

function     TPLine.MetamorfoseToREGION : TRegion;
begin
  Result:= TRegion.CreateFromCoords(FData);
  FData.clear;
  Free;
end;

//-----------------------------------------------------------------------------------------------------
// CLASSE REGION
//
//-----------------------------------------------------------------------------------------------------

class function TRegion.name;
begin
  result:= tagREGION;
end;

constructor  TRegion.create;
begin
  inherited;

  FData:= TStringList.Create;
  FSectionBounds:= TList.Create;
end;

constructor  TRegion.CreateFromCoords(const coords : TStringList);
begin
  FSections:= coords.count;
  FData:= TStringList.Create;
  FData.AddStrings(coords);

  FSectionBounds:= TList.Create;
//ATENÇÃO
//criar todos os bounds, ou melhor atribuir !!!
end;

procedure    TRegion.AssignCoordinates(origin : TCustomLayerItem);
var
   region : TRegion absolute origin;
begin
  inherited;
  FCenter:= region.center;
end;

procedure    TRegion.writeListObjectHeader;
var
   s : string;
begin
  s:= Format('%s %d',[name,FSections]);
  lines.append(s);
end;

procedure TRegion.RotateCoordinates; //(section, newFirst: integer);
var
```

```
   novoVet, vetAux : ^TCoordinate;
   n,size : integer;
begin
  n:= strToInt(FData[section-1]);
  size:= n * sizeof(TCoordinate);
  GetMem(novoVet, size);
  FCoordinates:= PArrayOfCoordinates(FData.Objects[section-1]);
// pego os N últimos e passo para a frente
  vetAux:= Pointer(FCoordinates);
  inc(vetAux, newFirst-1);
  Move(vetAux^, novoVet^, (n - newFirst + 1) * sizeOf(TCoordinate));
// pego o M-N primeiros e passo para trás
  vetAux:= novoVet;
  inc(vetAux, n - newFirst + 1);
  Move(FCoordinates^, vetAux^, (newFirst - 1) * sizeOf(TCoordinate));
// devolvo para o vetor original
  Move(novoVet^,FCoordinates^,size);
  {}
  freeMem(novoVet, size);
end;

//-----------------------------------------------------------------------------------------------
// CLASSE TEXT
//
//-----------------------------------------------------------------------------------------------

class Function TText.Name;
begin
  result:= 'TEXT';
end;

constructor  TText.create;
begin
//  system.delete(tag,1,Pos(' ',tag));
//  FText:= tag;
  FCoordCount:= 2;
  GetMem(FCoordinates,sizeOf(TCoordinate)*2);
end;

function     TText.getSize;
begin
  result:= FText;
end;

function     TText.getCoordinate(i : integer) : TCoordinate;
begin
  result:= FCoordinates^[i];
end;

procedure    TText.WriteToList;
begin
  with lines do begin
    append(name+' '+FText);
    append(format(' %s %s',[CoordinateToStr(FCoordinates^[1]),CoordinateToStr(FCoordinates^[2])]));
  end;
end;

// 8<-------------------------------------------------------------------------
// Implementação das classes de objects utilizadas dentro do SIAM
//
// 8<-------------------------------------------------------------------------

//DETT 21/01/1999
// A variável pontos é utilizada por todas as rotinas de desenho que trabalham
// com múltiplos pontos de uma só vez, note-se que estou utilizando um array
// estático para evitar alocações de memória
```

```
constructor   TCustomNode.createWithLayer;
begin
  inherited;
  FCoordCount:= 1;
  GetMem(FCoordinates,SizeOf(TCoordinate));
end;

function     TCustomNode.GetIDCode: string;
begin
  result:= FIdCode;
end;

function TCustomNode.GetIDLabel: string;
begin
  result:= FIdLabel;
end;

procedure    TCustomNode.SetIDCode(cod: string);
begin
  FIdCode:= cod;
end;

//-----------------------------------------------------------------------------------------------------

procedure   TCustomLine.SetCoordinate;
begin
  if (i > 0) and (i <= SectionDeep[j]) then    //range check na linha
    if (j > 0) and (j <= FSections) then begin //range check na coluna
       FCoordinates:= Pointer(FData.objects[j-1]);
       FCoordinates^[i]:= c;
    end;
end;

constructor TCustomLine.Create;
begin
//sectionCount: 0 : line; 1 : polyline with one section; n > 1 : multisection polyline
  if sectionCount = 0
    then begin
      //FSections:= 1;
      FData:= TStringList.Create;
      FSectionBounds:= TList.Create;
      //GetMem(FCoordinates,SizeOf(TCoordinate)*2);
      //FData.addObject('2',Pointer(FCoordinates));
      //system.delete(Tag,1,Length(tagLine)+1);
      //FCoordinates^[1].lon:= GetCoordinate;
      //FCoordinates^[1].lat:= GetCoordinate;
      //FCoordinates^[2].lon:= GetCoordinate;
      //FCoordinates^[2].lat:= GetCoordinate;
      //with FBounds do begin
      //  Left:= FCoordinates^[1].lon;
      //  Top:= FCoordinates^[1].lat;
      //  Right:= FCoordinates^[2].lon;
      //  Bottom:= FCoordinates^[2].lat;
      //end;
    end
    else inherited;
end;

constructor TCustomLine.createWithLayer;
begin
  inherited;
  FData:= TStringList.Create;
  FSectionBounds:= TList.Create;
end;

function TCustomLine.GetIDCode: string;
begin
```

```
  result:= intToStr(FIdCode);
end;

procedure TCustomLine.SetIDCode(cod: string);
begin
  FIdCode:= strToInt64(cod);
end;

procedure TCustomLine.SetSizes;
var
   i : integer;
begin
  KillDataItems(FData);
  FSections:= high(deeps) + 1;
  FDeepest:= 0;
  for i:= 0 to FSections-1 do begin
    GetMem(FCoordinates,sizeOf(TCoordinate)*deeps[i]); //aloco memória para as coordenadas
    FData.addObject(intToStr(deeps[i]),Pointer(FCoordinates));
    if deeps[i] > FDeepest then FDeepest:= deeps[i];
  end;
end;


// 8<---------------------------------------------------------------------------

constructor TCustomArea.createWithLayer;
begin
  inherited;
  FData:= TStringList.Create;
  FSectionBounds:= TList.Create;
end;

function   TCustomArea.GetIDCode;
begin
  result:= FIdCode;
end;

procedure  TCustomArea.SetIDCode;
begin
  FIdCode:= cod;
end;

function    TCustomArea.GetIDLabel;
begin
  result:= FIDLabel;
end;

procedure   TCustomArea.SetIDLabel;
begin
  FIDLabel:= cod;
end;

procedure TCustomArea.WriteBounds;
begin
  if s = 0
    then writeRectangle(arq,FBounds,true)
    else writeRectangle(arq,SectionBounds[s],true);
end;

{ TRectangle }

constructor TRectangle.create;
begin
  FSections:= 1;
  FData:= TStringList.Create;
  FSectionBounds:= TList.Create;
end;
```

```
class function TRectangle.Name: String;
begin
  result:= tagRectangle;
end;

procedure TRectangle.writeListObjectHeader(const lines: TStrings);
var
   s : string;
begin
  inherited;
  s:= format('%s %s %s',[name,CoordinateToStr(FCoordinates^[1]),CoordinateToStr(FCoordinates^[2])]);
  lines.append(s);
end;

procedure TCustomNode.SetIDLabel(cod: string);
begin
  FIdLabel:= cod;
end;

procedure TCustomLine.SetIDLabel(cod: string);
begin
  FIDLabel:= cod;
end;

function TCustomLine.GetIDLabel: string;
begin
  result:= FIDLabel;
end;

end.
```

ANEXO 2 – Artigo

Um conversor de dados geográficos, abrangendo os formatos MID-MIF, SHP-DBF, SDL.

André Demboski Pinter, Armando Luiz Dettmer

Ciências da Computação, 2003
Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-900
Fone (0XX48) 331-7513
pinter@inf.ufsc.br, armando@eps.ufsc.br

Resumo

No contexto de Sistemas de Informações Geográficas, percebe-se a necessidade do intercâmbio de dados no uso de aplicações de SIG. Entretanto, as dificuldades em se realizar este intercâmbio tornam a interoperabilidade um desafio a ser contornado pelos usuários desses sistemas. Visando a amenização deste problema, foi desenvolvido um conversor multidirecional entre os formatos MID-MIF, SHP-DBF e SDL, a fim de que a interação entre estes formatos fique transparente para os sistemas que os utilizam.

Palavras-Chave: Sistemas de Informações Geográficas, SIG, GIS, Conversor, MapInfo, ESRI, MapGuide.

Abstract

In the context of Geographic Information Systems, we realize the necessity of data interchange, when using applications of GIS. However, the difficulties involved in this interchange make the interoperability a challenge to be resolved for the users of that systems. In order to reduce this problem, a multidirecional translator was developed to MID-MIF, SHP-DBF and SDL formats, aspiring a transparent interaction of systems that use this formats.

Key-words: Geographic Information Systems, SIG, GIS, Converter, Translator, MapInfo, ESRI, MapGuide.

Introdução

O intercâmbio de dados espaciais é um importante desafio no uso das geotecnologias, impulsionado principalmente pelo alto custo de produção deste tipo de dados. Aliado a este fato, a maior parte dos produtos para suporte de SIG não são desenvolvidos para permitir a comunicação com outros sistemas computacionais. Genericamente, eles oferecem um ambiente de trabalho e uma linguagem para interação com o sistema próprios.

Para modelar objetos e fenômenos georreferenciados, cada SIG utiliza um modelo conceitual próprio e esta diversidade faz com que organizações produtoras de informação georreferenciada sigam regras conceituais vinculadas ao sistema por elas utilizado. O resultado é um ambiente heterogêneo, onde cada organização tem sua maneira de tratar a informação espacial [4].

Unindo os problemas citados a outros não menos expressivos, pode-se dizer que uma solução típica para resolvê-los consiste na tradução entre os dados disponíveis em geral para dados que possam ser manipulados pela aplicação desejada. Em geral, este é o objetivo principal do trabalho: poder converter dados geográficos de formatos distintos, de forma bidirecional, a fim de torná-los compatíveis ao sistema que se deseja.

SIG

Na era da informação como grande riqueza do nosso tempo, o nível de qualidade desta informação deve ser o mais substancial possível. Nas últimas décadas, diversas tecnologias apareceram para auxiliar na solução do problema acima. Entre estas, a aplicação da referência geográfica da informação em sistemas computacionais – os Sistemas de Informação Geográfica (SIGs) – possibilita dar uma melhor visualização do problema, facilitando a tomada de decisões [2].

O termo SIG vem sendo objeto de várias definições por parte de diferentes autores. Porém, podemos dizer que fundamentalmente o termo SIG pode ser definido de duas maneiras distintas. A primeira diz respeito à utilização para referir de forma genérica a um sistema de informação que contempla características relativas a localizações espaciais. Já a segunda utiliza o termo para referir um tipo determinado de produto comerciai, especialmente direcionado para a realização de sistemas que envolvem dados representando localizações geográficas.

Efetivamente, a informação geográfica organizada por temas, tem sido tradicionalmente apresentada sob a forma de mapas desde as mais antigas civilizações. Recorrendo apenas a processos manuais, foi possível representar em folhas de papel o resultado das observações efetuadas sobre algumas características da superfície terrestre. No entanto, é perceptível que tais mapas apresentavam diversas limitações, devido ao caráter manual que lhes era inerente.

Desta forma, o recurso a meios computacionais para suporte de informação espacial iniciou-se no princípio da década de 60, com a codificação digital da informação que, tradicionalmente, apenas era representada sob a forma de mapas. Contudo, só os avanços no campo da tecnologia informática alcançados no início da década de 70, particularmente os relacionados com o acesso direto a discos, permitiram obter resultados significativos.

Atualmente, os SIG são utilizados como ferramentas de análise geográfica, por excelência, já que permitem a integração de grandes volumes de informação espacial e de outros tipos num mesmo sistema e o seu tratamento conjunto. Esta integração tornou-se possível como resultado da convergência de várias disciplinas e técnicas tradicionais, conforme a figura seguinte [1]:

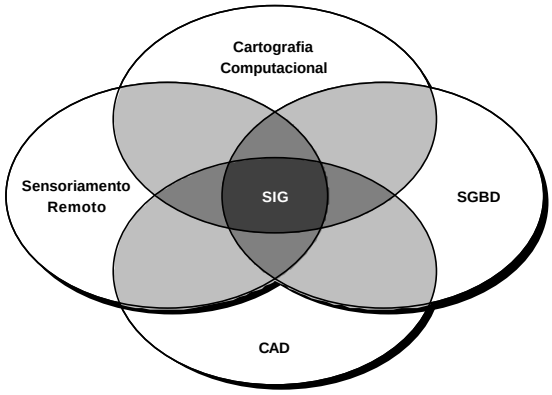

Fig. 1 – Origem dos SIGs.

O termo Sistema de Informação Geográfica (SIG) é aplicado para sistemas que realizam o tratamento computacional de dados geográficos. Devido à sua ampla gama de aplicações, que inclui temas como agricultura, florestas, cartografia, cadastro urbano e redes de concessionária (água, energia e telefonia), há pelo menos três formas de se utilizar um SIG:

- como ferramenta para produção de mapas;
- como suporte para análise espacial de fenômenos e
- como um banco de dados geográficos, com funções de armazenamento e recuperação de informação espacial.

Sabendo das diferentes definições de SIG, pode-se dizer que elas expressam visões de diferentes pesquisadores, porém convergem para um mesmo ponto. Integrar em um único banco de dados informações espaciais de diferentes áreas, oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, e para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

Interoperabilidade

Os SIGs têm sido bastante utilizados a partir dos anos 80 como suporte para a tomada de decisões em áreas como administração pública, meio ambiente, marketing, etc. Cada produto de software comercial de SIG se desenvolveu independentemente, com poucas terminologias e teorias em comum. Como resultado, é bastante complicado para diferentes sistemas compartilharem dados, para usuários treinados em um sistema utilizarem outro, ou para compartilhar

aplicativos desenvolvidos em diferentes sistemas. O termo interoperabilidade sugere um mundo ideal onde não existiriam estes problemas, ou pelo menos onde estes fossem minimizados.

Veremos então, a seguir, algumas abordagens para esta interoperabilidade aqui referida, como também suas características e problemas.

Conversão Sintática

A abordagem mais básica para a interoperabilidade entre SIGs é a conversão sintática direta, que faz a tradução de arquivos de dados geográficos de um formato para o outro. Para que seja permitido esse tipo de conversão os SIGs trabalham com duas alternativas:

- oferecer um formato de exportação ASCII de fácil legibilidade, como DXF (Autocad), MID/MIF (MapInfo), E00 (Arc/Info) e SPR (Spring);
- documentar as estruturas de dados internas, como no caso do SHP (ArcView).

Porém, a utilização destes formatos não exclui a possibilidade da ocorrência de distorções nos dados, quando da transferência das informações, pois eles são organizados de acordo com o sistema gerador e quando importados para sistemas conceitualmente diferentes, necessitam de manipulação externa.

Apesar das limitações da conversão sintática, deve-se reconhecer que a grande maioria dos processos de conversão de dados opera neste nível, assim como o conversor proposto por este trabalho, porém com algumas melhorias trazidas de outras abordagens.

Metadados

Metadados descrevem o conteúdo, condição, histórico, a localização e o formato do dado. O objetivo do seu uso é ter um mecanismo para identificar qual dado existe, a sua qualidade e como acessá-lo e usá-lo.

A principal proposta de padrão de metadados é do FGDC (*Federal Geographic Data Committee*), comitê que promove a coordenação do uso, troca e disseminação de dados espaciais nos EUA. Este padrão ratifica um conjunto comum de definições para documentar o dado geográfico, incluindo: identificação, qualidade do dado, organização espacial do dado, referência espacial, informação sobre entidade e atributo, distribuição e referência do metadado [4].

Como sua ênfase é na disponibilidade da informação, o padrão FGDC não especifica a maneira pela qual a informação está organizada nem o processo de transferência. Com exceção da parte de entidades e atributos, que pode revelar parte do significado do dado, as demais partes não descrevem a semântica da informação espacial.

O grande problema da proposta do FGDC (e do uso de metadados em geral) é a excessiva ênfase em informações que descrevem o processo de produção dos dados. Isto quer dizer que, o esforço desprendido para a elaboração da documentação e preenchimento dos formulários, exigidos por esse padrão, não compensa em razão do que é obtido como resultado.

Em resumo, a substancial burocracia envolvida em adotar o padrão FGDC não se traduz em benefícios proporcionais. Estes fatos talvez expliquem porque sua adoção ainda está limitada e porque o consórcio OpenGIS, sobre o qual será falado mais adiante, propõe seu próprio formato para metadados.

Conversão Semântica

O aspecto semântico diz respeito à representação conceitual da informação geográfica presente em cada sistema. Como comunidades com cultura e história diferentes que interpretam distintamente a realidade geográfica e produzem sistemas conceitualmente heterogêneos, a capacidade de transferir dados de um sistema para outro não garante que os dados têm significado para o novo usuário [4].

Esta possível incompatibilidade aponta que, para que o intercâmbio aconteça, um conjunto consistente de interpretações deve estar disponível para a informação, levando a um significado comum sobre o dado trocado.

Analisando semanticamente, uma plena interoperabilidade entre diferentes formatos depende de compartilhamento de conceitos comuns entre os membros de uma comunidade de informação. Estão incluídos nestes conceitos: o modelo de dados; o dicionário de conceitos, indicado por uma ontologia comum; o dicionário de procedimentos, que contém os diferentes mecanismos de consulta, manipulação e apresentação utilizados para extrair informação dos dados compartilhados.

O uso de Ontologias para interoperabilidade de dados geográficos ainda está nos seus primórdios, sem exemplos concretos nem padrões estabelecidos. Espera-se que, nos próximos anos, haja um substancial desenvolvimento neste campo [4].

XML

Além dos usuários de SIG, o problema da interoperabilidade atinge também usuários de

outros sistemas computacionais, impulsionado principalmente pelo avanço das redes de computadores. O padrão XML surgiu, justamente, com o propósito de resolver este problema.

De acordo com a proposta de XML, o conteúdo de um documento (ou seus dados) é que devem ser codificados, mas não a sua forma de apresentação. A aplicação que utilizará estes dados é que deve ser responsável por interpretá-los e apresentá-los ao usuário.

O uso de XML é vantajoso, pois é ideal para descrever estruturas de dados hierárquicas e complexas, características comuns em dados espaciais. O dado permanece legível, pois usa a codificação ASCII, e acessível através de programas que acessam dados no padrão, os "*parsers* XML". Além destas vantagens, o uso de XML vem aumentando e este cada vez mais afirmado como padrão para armazenar e trocar dados [4].

A utilização de XML para atuar no problema da interoperabilidade entre SIG foi uma proposta do consórcio OpenGIS, que resultou na publicação, em 2000, do *paper* de recomendação da Geographic Markup Language (GML) 1.0, baseada na tecnologia XML.

GML

GML foi especificada para o transporte e armazenamento de informação geográfica, incluindo propriedades espaciais e não espaciais das feições geográficas. Esta linguagem define, atualmente, três esquemas:

- *Feature.xsd* – Define tipos e elementos concretos e abstratos de acordo com a especificação OpenGIS;
- *Geometry.xsd* – Define a geometria de acordo com a especificação OpenGIS;
- *Xlinks.xsd* – Define formas de ligação entre documentos e elementos dentro de um documento XML.

Seguindo os princípios da linguagem em questão, estes esquemas podem ser estendidos para criar outros esquemas XML para aplicações específicas. Assim, prevalece o que a tecnologia XML propõe que é a possibilidade de estender os modelos, criando novas "*tags*", e cada instituição pode criar seus esquemas ou simplesmente utilizar esquemas de terceiros em seus SIGs, promovendo assim, a tão desejada interoperabilidade [SILVA].

A utilização de esquemas XML oferece algumas facilidades na construção do código GML como, promover a utilização de elementos de esquemas diferentes para construir esquemas personalizados utilizando namespaces, permitir o controle sobre estruturas hierárquicas e possibilitar uma maior flexibilidade na definição de tipos de dados.

Dentre os principais objetivos da GML estão:

- Prover uma estrutura aberta para a definição de esquemas de aplicações e objetos espaciais;
- Permitir perfis que suportem subconjuntos das capacidades declarativas de GML;
- Suportar a descrição de esquemas de aplicações geoespaciais para domínios específicos e comunidades de informação;
- Permitir a criação e manutenção de esquemas e dados geoespaciais distribuídos e relacionados;
- Aumentar a habilidade das organizações para compartilhar dados geográficos;
- Servir como ferramenta para interoperabilidade de Sistemas de Informação Geográfica de uma maneira incremental [8].

A utilização de GML se baseia em conceitos que são comuns no contexto de SIG, como pontos linhas e polígonos e, por este fato, também apresenta deficiências nas questões semânticas.

Outro empecilho à sua utilização é a disponibilidade de ferramentas computacionais adequadas, uma vez que as ferramentas disponíveis não comportam as manipulações desejadas de dados e esquemas GML, ora permitindo só leitura, ora escrita. E quando leitura e escrita são permitidas não se permite manipulação de esquemas de aplicação.

Padrão OpenGIS

A OGC (*Open GIS Consortium)* é uma organização internacional que está criando novas padronizações técnicas e comerciais para garantir interoperabilidade em SIG. Fundada em 1994 por fornecedores de software, companhias de telecomunicações, universidades, provedores de informação e órgãos governamentais, entre outros, a OGC busca criar uma especificação de software e novas estratégias empresariais, a fim de tornar os sistemas de geoprocessamento abertos e integrar completamente os dados geográficos e as operações necessárias para manipulá-los.

Esta união de forças entre produtores de software, fabricantes de hardware, instituições de pesquisa e entidades governamentais é a alavanca que faz esta abordagem ganhar força no mercado em geral. Afinal, o fato de os diversos padrões de intercâmbio adotados em alguns países desenvolvidos não se firmarem como padrões mundiais é devido, certamente, ao caráter individual que possuem.

O padrão OpenGIS está em fase de desenvolvimento e está tentando reunir esforços para que a grande heterogeneidade dos produtos de software de SIGs possam conversar entre si, e além disso, que as instituições possam trocar informações espaciais entre si, através de um embrião de uma proposta de Comunidade de Informação Espacial, que trata também de padronização dos procedimentos organizacionais das instituições [2].

Apesar de inegáveis avanços que a proposta OpenGIS apresenta no campo da interoperabilidade, ela ainda possui várias limitações. Entre elas, podemos citar que a existência de uma API resolve apenas o problema de acesso padronizado a bancos de dados espaciais e não substitui a necessidade da transferência dos dados entre sistemas. Isto se deve talvez ao fato desta proposta ter sido colocada em desenvolvimento com um certo atraso, o que fez com que SIGs com dados de estruturas diferentes se disseminassem pelo mercado, surgindo assim o problema da interoperabilidade.

Formatos de Dados

Genericamente falando, pode-se dizer que a definição de padrões facilita o compartilhamento, a integração e a transferência de dados. Quando abordamos padrões para SIG, usualmente estamos nos referindo a padrões para linguagem de especificação, transferência de dados, geocodificação, documentação de metadados e de formatos.

Tendo em vista o crescimento dos sistemas de informação geográfica disponíveis, vários formatos de intercâmbio foram surgindo para atender as necessidades dos clientes. Assim, os tópicos a seguir apresentam de forma resumida as principais características dos formatos abordados neste trabalho.

MID-MIF

O formato de intercâmbio MapInfo Interchange File (MIF) teve origem no lançamento da primeira versão do software MapInfo, pela empresa norte americana MapInfo Corporation, em 1994. Este formato de intercâmbio permite a leitura de arquivos MIF por outros SIG existentes [6].

O MIF é um arquivo texto ASCII que descreve o conteúdo de uma tabela MapInfo e consiste, na verdade, de dois arquivos relacionados: um que armazena os dados gráficos e outro que armazena dados tabulares. Os dados gráficos estão em um arquivo com a extensão **.mif**, enquanto os dados tabulares estão no outro arquivo com a extensão **.mid**. Estes arquivos podem ser lidos e escritos pelo software MapInfo Professional e traduzidos para outros formatos através de outros programas.

O arquivo MIF é composto por duas áreas – a área de cabeçalho do arquivo e a seção de dados – que seguem uma especificação bem detalhada para a sua construção. A seção de dados, que vem logo após o cabeçalho, pode conter diversas primitivas gráficas, uma para cada objeto gráfico. O MapInfo estabelece uma correspondência entre cada objeto especificado no arquivo MIF para cada linha de dado no arquivo MID, associando o primeiro objeto no arquivo MIF com a primeira linha no arquivo MID, o segundo objeto com a segunda linha e assim sucessivamente.

Quando da não existência de um objeto gráfico correspondente a uma linha particular no arquivo MID, um "objeto em branco" (NONE) deve ser escrito no arquivo MIF.

Desta forma, os seguintes objetos podem ser especificados:

- ponto (point)
- linha (line)
- poligonal (poly-line)
- região (region)
- arco (arc)
- texto (text)
- retângulo (rectangle)
- retângulo arredondado (rounded rect.)
- elipse (ellipse)

O arquivo MID é um arquivo texto que tem seus dados delimitados de tal forma que cada linha corresponde a um registro de dados. Além disso, cada linha está associada a um objeto correspondente no arquivo MIF, que contém também a descrição dos campos em sua primeira parte. Caso o objeto de desenho no MIF não tenha um registro correspondente, deve ser colocada, no MID, uma linha com os campos em branco. O MID é um arquivo opcional, quando ele não é utilizado, todos os seus campos ficam em branco.

SHP-DBF

Um arquivo Shapefile (SHP) armazena, de forma não-topológica, características espaciais de geometria e atributos, em um conjunto de dados. As características de geometria são armazenadas num formato composto por um conjunto de coordenadas vetoriais.

Devido ao fato de não haver o excesso de processamento de uma estrutura de dados

topológica, estes arquivos levam algumas vantagens em relação a outras fontes de dados, como velocidade de desenho e capacidade de edição mais rápidos. Além disso, os arquivos SHP geralmente necessitam de menos espaço em disco e são mais fáceis de ler e escrever.

Na verdade, um shapefile consiste de um arquivo principal, um arquivo de índice e uma tabela dBASE. O principal é um arquivo de registros variáveis, onde cada registro descreve uma informação geográfica, através da lista de seus vértices. No arquivo de índice, cada registro contém a posição do registro correspondente no arquivo principal, sempre em relação ao início do arquivo. Já a tabela dBASE contém características de atributos, sendo um registro por característica. Esta relação de um para um entre geometria e atributos é baseada no número de registros, ou seja, registros em arquivos dBASE devem estar na mesma ordem que os registros no arquivo principal.

Como forma de relacionar os arquivos que descrevem uma informação geográfica, devemos nomeá-los de forma igual, apenas diferenciando suas extensões. O arquivo principal, o arquivo de índice e a tabela dBASE possuem, respectivamente, extensões **.shp**, **.shx**, **.dbf**.

Neste formato, pode-se especificar os seguintes objetos:

- ponto (point)
- multiponto (multipoint)
- poligonal (polyline)
- polígono (polygon)

Um simples arquivo shp pode conter diversos objetos de desenho, porém, todos eles têm que ser do mesmo tipo (arquivo só de linhas, ou só de pontos, etc).

SDL

O formato Spatial Data Load (SDL) consiste de um arquivo no formato ASCII utilizado pelo software Autodesk MapGuide e também por outras ferramentas de manipulação de mapas na World Wide Web.

A representação dos dados em SDL pode ser feita tanto em duas dimensões como também em três dimensões (2D e 3D). Este tipo de arquivo armazena ambas as informações, de geometria e de atribuição. Em tempo, um conjunto lógico de dados em SDL consiste de um ou mais arquivos contidos em um mesmo diretório, todos eles possuindo a extensão **.sdl**.

As representações geométricas podem ser dos seguintes tipos:

- ponto (point)
- linha (line)
- poligonal (polyline)
- polígono (polygon)

Entretanto, cada arquivo SDL deve conter apenas um tipo geométrico. É permitido também que sejam armazenadas informações sem características geométricas. Além disso, os tipos de objetos podem ser múltiplos. Desta forma, um objeto ponto pode ser formado, na verdade, por vários pontos, um objeto linha ser formado por várias linhas e assim por diante.

Conversor de Dados Geográficos

O conversor de dados geográficos desenvolvido neste trabalho permite o intercâmbio dos formatos MID-MIF, SHP-DBF e SDL, minimizando o máximo possível as distorções, a fim de que sejam imperceptíveis ao usuário de SIG. Adicionalmente, este será de suma importância para o Laboratório de Transportes, uma vez que possibilitará a conversão de diversas bases de dados disponíveis na Internet para os formatos utilizados por suas aplicações.

Internamente, a estrutura do conversor consiste de diversos parsers, um para cada formato de arquivo, que têm a responsabilidade de manipular os dados de seus respectivos formatos. Desta forma, a extensibilidade do conversor para a manipulação de outros formatos geográficos se torna mais fácil, pois basta, praticamente, desenvolver o parser específico para um novo formato em questão.

Como foi visto anteriormente, os formatos abrangidos pelo conversor possuem objetos de desenho particulares, sendo diferentes de formato para formato. Assim, para uma melhor compreensão de como é feito o mapeamento dos objetos ente os diferentes formatos, exemplifica-se uma conversão hipotética:

Um usuário, possuindo um arquivo MIF de linhas, deseja convertê-lo para o formato SHP. Desta forma, o parser de MIF se encarrega da leitura do arquivo inicial, as *lines*, e do mapeamento para o formato interno do conversor, que neste caso também são *lines*. Tendo feito isso, o parser de SHP se encarrega de, a partir no formato interno, fazer o mapeamento para *polylines*, armazenando os dados manipulados em um arquivo diferente.

Esta lógica é estendida para todos os outros objetos de desenho, de acordo com a tabela 1:

Tipos de Objeto	Interno	MIF	SHP	SDL
Ponto	TCustomNode	Point	Point	Point
Linha	TCustomLine	Line	Polyline	Line
Poligonal	TCustomLine	Polyline	Polyline	Polyline
Região	TCustomArea	Region	Polygon	Polygon
Arco	*	Arc	-	-
Texto	*	Text	-	-
Retângulo	TCustomArea	Rectangle	Polygon	Polygon
Retângulo arredondado	TCustomArea	Rounded rectangle	-	-
Elipse	-	Ellipse	-	-

Tabela 1 – Mapeamento dos objetos de desenho no Conversor.

Como mencionado anteriormente, o conversor é composto por diversos parsers, responsáveis por manipular os dados do formato a que eles se referem. Assim, podemos dizer que os parsers possuem, de forma genérica, estruturas bem semelhantes, pois devem ler os dados de seu formato e convertê-los para o formato interno, assim como também a operação inversa, receber dados no formato interno e convertê-los para seu formato. A estrutura do funcionamento de um parser pode ser melhor visualizada na figura 2:

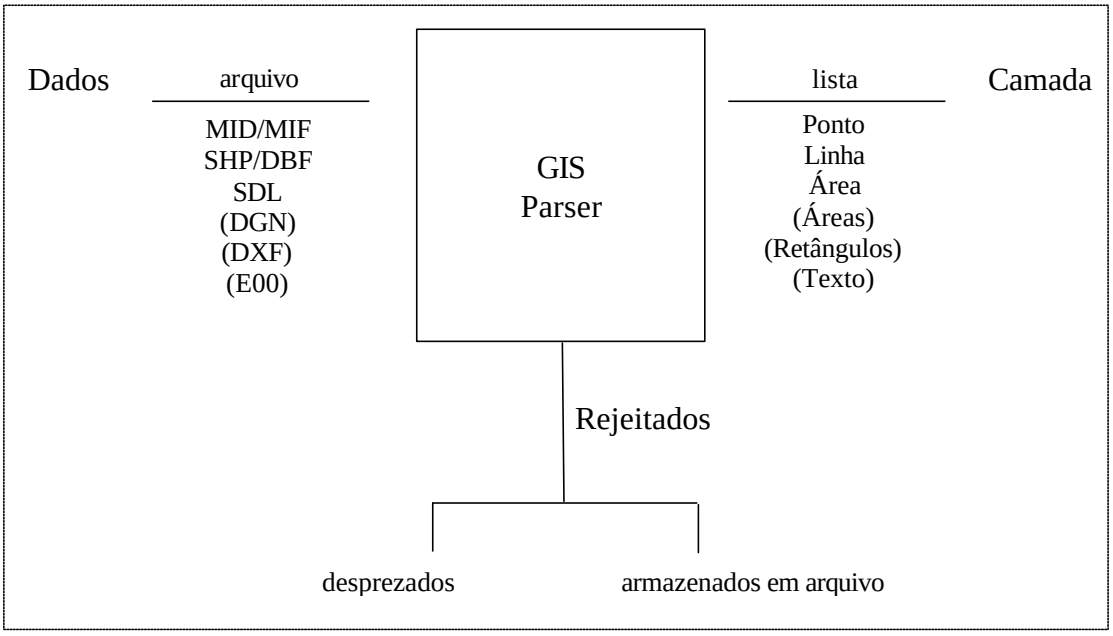

Fig. 2 – Estrutura interna do Conversor.

Em um processo de conversão, os objetos de desenho são lidos a partir do arquivo de origem, pelo parser do formato a qual este arquivo pertence. No momento desta leitura, cada objeto é identificado (ponto, linha, área, etc...) e é conferida a sua possibilidade de mapeamento para o formato interno do conversor. Em caso positivo, cria-se uma nova instância do objeto interno adicionando-o como elemento de uma camada, sendo este processo repetido até que todos os objetos do arquivo tenham sido inseridos na camada. Após a inserção de todos os objetos contidos no arquivo, esta camada, que também foi criada em tempo de execução, é adicionada a

uma lista de camadas, que é interna ao conversor.

Desta forma, para cada arquivo de origem, é criada uma camada interna, e a partir de cada camada, pode-se fazer o processo inverso, gerando-se arquivos de destino no formato desejado.

Conclusões

Durante a elaboração do trabalho, observou-se que o termo Sistemas de Informação Geográfica (SIG) pode ter diversas definições, de acordo com os diferentes autores da comunidade científica. Entretanto, o foco de sua aplicação é um fator que está bem claro e definido, que é integrar as informações em um único banco de dados e oferecer mecanismos de análise e manipulação sobre elas.

Mesmo com as diversas abordagens de interoperabilidade citadas no trabalho, percebeu-se que fazer esta integração é uma tarefa muito difícil, e que certas vezes acarreta em perda de informação para o usuário.

Analisando o intuito do conversor de dados desenvolvido neste trabalho, pode-se afirmar que ele auxilia a integração de SIGs distintos, pois permite fazer as conversões de dados necessárias para que estes sistemas interajam. Este auxílio se dá principalmente ao Laboratório de Transportes que, de posse do conversor, pode agora disponibilizar às suas aplicações, dados geográficos antigamente não importados por elas.

Entretanto, ainda há um longo caminho a ser percorrido para que o conversor possa solver problemas de interoperabilidade no âmbito geral, encobrindo diversos formatos nas diversas áreas de interesse.

Referências

[1] ABRANTES, G. Sistemas de Informação Geográfica – Conceitos. 1998. Disponível em: http://www.isa.utl.pt/dm/sig/sig/SIGconceitos.html. Acesso em: 17 dez. 2003.

[2] ALMEIDA, M. A. Análise da interoperabilidade em sistemas de informação geográfica. **Ratio: revista do Instituto Luterano de Ensino Superior de Ji-Paraná** – Revista da Faculdade ULBRA. Ji-Paraná – MT, n. 5, p. 3-9. Jun, 2002.

[3] ESRI. ESRI Shape File Technical Description. Jul, 1998. Disponível em: www.esri.com/library/whitepapers/pdfs/shapefile.pdf. Acesso em: 18 dez 2003.

[4] LIMA, P.; CÂMARA, G.; QUEIROZ, G. GeoBR: Intercâmbio Sintático e Semântico de Dados Espaciais. In: Simpósio Brasileiro de Geoinformática. IV. 2002. Caxambu – MG. Anais GEOINFO 2002. P. 139 – 146.

[5] MAPINFO COMPORATION. The MapInfo Interchange File (MIF) Format Specification. Out, 1999. Disponível em: http://ees2.geo.rpi.edu/gis/data/mapinfo_mif.pdf. Acesso em: 18 dez 2003.

[6] MONTEIRO, R. P. **Intercâmbio de dados entre Sistemas de Informação Geográficos.** 2000. Monografia (Especialização em Informática Pública). PUC – MG.

[7] SAFE SOFTWARE. Feature Manipulation Engine (FME) Readers and Writers. Cap. 56. Disponível em: ftp://ftp.safe.com/fme/2004/docs/ReadersWriters2004.zip. Acesso em: 23 jan 2004.

[8] SILVA, P. P. O. da. **Codificação XML para armazenamento e transporte de dados geográficos.** 2002. Trabalho acadêmico (Mestrado em Informática). UFRJ. Rio de Janeiro – RJ.

[9] THOMÉ, R. **Interoperabilidade em geoprocessamento: conversão entre modelos conceituais de Sistemas de Informação Geográfica e comparação com o padrão OpenGIS.** 1998. Dissertação (Mestrado em Computação Aplicada). INPE. São José dos Campos – SP.

[10] XAVIER, C. C. Sistemas de Informação Geográficas – Trabalhos em desenvolvimento pelo IMPA e INPE. Disponível em: http://orion.lcg.ufrj.br/seminarios/gis.ppt. Acesso em: 28 dez 2003.