UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Conjunto de Procedimentos de Engenharia Reversa para Projeto de Banco de Dados Espaciais

por

LIA CLÁUDIA MATTÉ

Trabalho de Conclusão submetido como requisito parcial
para a obtenção do grau de Mestre em Informática

Prof. Dr. Cirano Iochpe
Orientador

Porto Alegre, novembro de 2002.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Matté, Lia Cláudia

Conjunto de Procedimentos de Engenharia Reversa para Projeto de Banco de Dados Espaciais / por Lia Cláudia Matté - Porto Alegre: PPGC da UFRGS, 2002.

107 f.:il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós Graduação em Computação, Porto Alegre, RS – Brasil, 2002. Orientador: Iochpe, Cirano.

1. Sistemas de Informação Geográfica. 2. Engenharia Reversa. 3. GeoFrame. I. Iochpe, Cirano. II. Título.

Agradecimentos

Agradeço ao meu orientador, Prof. Cirano Iochpe, pela confiança, apoio técnico e ensinamentos transmitidos durante o desenvolvimento do trabalho.

Às universidades de Caxias do Sul e Federal do Rio Grande do Sul, por criarem o convênio que permitiu a realização deste trabalho.

Ao Projeto SIME, por disponibilizar informações indispensáveis para a conclusão deste trabalho.

À minha família, em especial à minha mãe, pelo amor que sempre me foi dedicado.

Agradeço a grande amiga Helen Zatti, pelas inúmeras caronas Caxias-Porto Alegre e pelas palavras de incentivo.

Finalmente, agradeço a Deus por tudo.

Sumário

Lista de Abreviaturas

BNF	Forma Backus-Naur
CSN	Class Syntax Notation
CUT	Coordenada Universal do Tempo
GML	Geography Markup Language
GPS	Global Positioning Systems
ISO	International Standard Organization
OSN	Object Syntax Notation
SAIF	Spatial Archive and Interchange Format
SDTS	Spatial Data Transfer Standard
SGBD	Sistema Gerenciador de Banco de Dados
SIG	Sistemas de Informação Geográfica
SIME	Sistema de Informações Georreferenciadas da Região Metropolitana de Belém
SRE	Sistema de referência espacial
UML	Unified Modeling Language
URL	Uniform Resource Locator
XML	Extensible Markup Language

Lista de Figuras

Lista de Tabelas

Resumo

Projetos de SIG (Sistemas de Informação Geográfica), em geral, ainda não apresentam modelo conceitual de banco de dados geográficos. Os implementadores de SIG se preocupam com a aquisição dos dados (captura e adaptação), que é a etapa mais cara, e dão, ainda, pouca atenção à modelagem. A utilização de modelos conceituais faz com que os usuários controlem melhor sua base de dados e tirem maior proveito do SIG. O objetivo deste trabalho é definir um conjunto de procedimentos de engenharia reversa de bancos de dados espaciais que auxiliem na criação e manutenção de modelos conceituais para aplicações de SIG a partir de dados já existentes. Estes procedimentos podem contribuir para um aumento na qualidade dos SIG implementados, auxiliando na popularização da prática de modelagem conceitual de banco de dados geográficos. São estudados três formatos de transferência utilizados por usuários de SIG. A partir daí, é proposto uma arquitetura de sistema de engenharia reversa para SIG.

Palavras-Chave: Sistemas de Informação Geográfica, Engenharia Reversa, GeoFrame, Modelo Conceitual.

TITLE: "SET OF REVERSE ENGINEERING PROCEDURES FOR SPATIAL DATABASE DESIGN"

Abstract

Even with the increasing use of Geographic Information Systems (GIS), conceptual modeling of geographic databases (GDB) is not yet usual among GIS professionals. This fact can in part be explained by the professional profile of most GIS designers. They usually are more familiar with specific GIS applications than with database technology. Among them, one can find cartographers, geographers, agricultural engineers, and architects. Besides that, the cost of geographic data acquisition is so high that much important is given to acquisition plans during GIS design and less interest is given to the database design process. Especially first time designers do not perceive the importance of a database schema that is independent of a specific GIS product. Conceptual design preserves the independence between data types and the logical schema of the product. Moreover, relying on the GDB conceptual schema users can better understand what part of the geographic reality is represented in the database. The main goal of this research work is to define a set of reverse engineering procedures for spatial databases that can support either creation or evolution if conceptual GDB schemas. It is expected that these procedures can enhance the quality of GDB design as well as contribute to the popularization of GDB conceptual modeling. Three different geographic data input formats to GIS are investigated and a reverse engineering system architecture for them is proposed. Resulting conceptual (sub)schemas are based on the framework GeoFrame.

Keywords: Geographic Information Systems (GIS), Geographic Database (GDB), Reverse Engineering, conceptual framework GeoFrame, Conceptual Design.

1 Introdução

1.1 Motivação

Sistemas de Informação Geográfica (SIG) possibilitam a análise e manipulação de dados georreferenciados, ou seja, dados que possuem referência geográfica em relação à superfície terrestre [WEB99]. Sistemas de informação geográfica necessitam gerenciar grande volume de dados, uma vez que a realidade por eles representada é bastante complexa. Eles estão, normalmente, associados a sistemas de gerência de banco de dados (SGDB).

Geralmente, cada empresa ou instituição produz seu próprio banco de dados com as informações específicas para sua aplicação. A coleta de dados é um processo bastante custoso e tornou-se a etapa mais longa e crítica das aplicações de SIG [WEB99]. Normalmente, os dados são obtidos através da digitalização de mapas, levantamento aerofotogramétrico, mapeamento de regiões através de GPS (*Global Positioning Systems*) e imagens de satélite de alta resolução.

A fim de minimizar os custos e facilitar a aquisição de dados, há grande interesse, atualmente, no intercâmbio de informações geográficas entre instituições. Várias soluções têm sido propostas para facilitar este intercâmbio, desde traduções entre formatos de dados, de um fabricante de *software* para outro, até especificações complexas de protocolos de dados e especificações de *software* para promover a interoperabilidade entre sistemas como, por exemplo, o SDTS (*Spatial Data Transfer Standard*) [ANS97] e o OpenGIS [OGC99], respectivamente.

Para que a troca de informações seja eficiente é necessário conhecer os dados disponíveis e a qualidade dos mesmos. Devido à falta de documentação apropriada (ex.: esquema conceitual de dados) e da freqüente aquisição de dados de terceiros, muitos usuários desconhecem as informações que têm armazenadas em suas bases de dados, o que limita sua potencialidade de uso e dificulta o intercâmbio. Um modelo conceitual dos dados e uma base de metadados, ambos atualizados, tornam-se importantes dentro desse contexto.

A utilização de técnicas de projeto de banco de dados já está amplamente comentada e consolidada na literatura [ELM94,HEU98]. A importância destas técnicas para SIG não é menor. Entretanto, a maioria dos projetistas que trabalham com SIG não faz uso das técnicas tradicionais de modelagem ou de outras que tenham sido desenvolvidas especificamente para a realidade das informações geográficas. Isso ocorre em decorrência de diversos fatores. Em [LIS2000] o autor cita:

- ✓ as aplicações de SIG são projetadas e desenvolvidas, muitas vezes, pelos próprios usuários finais, os quais, normalmente, possuem formação em diferentes áreas como, por exemplo, geografia, cartografia, biologia, agronomia e engenharia. Em geral, estes profissionais não estão familiarizados com técnicas de projeto de banco de dados, partindo direto para a criação do sistema na *interface* do produto de SIG;
- ✓ a inexperiência, desses projetistas, em relação ao uso de metodologias de desenvolvimento de sistemas de informação tem resultado na construção de sistemas com sérias limitações de uso (ex.: ocorrência freqüente de dados redundantes e, muitas vezes, inconsistentes); e

- ✓ o elevado custo do processo de aquisição e preparação de dados georeferenciados, conseqüência de sua característica espaço-temporal, faz com que o esforço do projetista esteja voltado, principalmente, para este problema.

Mecanismos que facilitem a criação e manutenção de modelos conceituais nas aplicações de SIG podem popularizar o uso dos mesmos pelos projetistas, aumentando a qualidade dos SIG implementados. Desta forma, o intercâmbio de dados entre instituições poderia ser incrementado, reduzindo os custos de obtenção de dados. Neste contexto, este trabalho propõe uma arquitetura de sistema de engenharia reversa para SIG capaz de inferir um modelo conceitual a partir de uma fonte de dados fornecida pelo usuário.

1.2 Trabalhos Relacionados

Em [HEU98], o autor define um processo de engenharia reversa como um processo de abstração, que parte de um modelo de implementação e resulta em um modelo conceitual que descreve abstratamente a implementação em questão.

No início da década de noventa a necessidade de reengenharia dos chamados sistemas legados motivou várias pesquisas sobre engenharia reversa. Recentemente, a demanda de diversos setores em adaptar seus sistemas de informação para o contexto da Web tem gerado a necessidade de novos métodos, ferramentas, e infraestrutura para explorar eficientemente, e a custos baixos, aplicações já existentes [MUL00].

As pesquisas sobre engenharia reversa desenvolveram-se nas áreas de engenharia de software e banco de dados, entre outros. Nos últimos dez anos, pesquisadores desenvolveram diversas formas de explorar, manipular, analisar, sintetizar, interligar, e visualizar artefatos de software. As principais pesquisas e publicações sobre o assunto concentram-se na definição de metodologias, ferramentas de engenharia reversa, aplicações específicas e em experiências de aplicações de engenharia reversa [DAV2000].

Em [HEU98], o autor apresenta um processo de engenharia reversa que permite obter um modelo lógico relacional a partir do modelo lógico de um banco de dados não relacional. O processo parte das descrições dos arquivos que compõem o sistema existente. Representa a descrição de cada arquivo na forma de um esquema de uma tabela relacional não-normalizada. A seguir, o esquema de tabela não-normalizada passa por um processo conhecido como normalização. Neste processo, as informações são reagrupadas, eliminando-se a redundância de dados. Depois de todos os arquivos do sistema terem sido normalizados, os diferentes esquemas relacionais obtidos são integrados, gerando o esquema relacional do banco de dados do sistema. Nesta etapa, as informações comuns a diferentes arquivos são identificadas e representadas uma única vez.

Uma metodologia genérica para engenharia reversa de banco de dados é apresentada em [HAI2002]. Esta metodologia divide o processo de engenharia reversa em duas fases: processo de extração da estrutura dos dados e processo de conceitualização da estrutura dos dados.

A primeira fase consiste na recuperação completa do esquema do sistema de armazenamento de dados, incluindo todas as estruturas implícitas e explícitas e regras de limitação. A complexidade desta fase depende do formato de armazenamento dos dados. Sistemas de bancos de dados geralmente fornecem alguma documentação sobre

seus esquemas, facilitando muito a análise de suas estruturas. Entretanto, se os dados forem oriundos de arquivos padrão, dificilmente existe alguma documentação sobre sua estrutura. Isso obriga, muitas vezes, a análise de código, execução de programas, análise do *layout* de telas e relatórios, etc. Os processos que compõem esta fase são os seguintes: análise textual das declarações das estruturas dos dados (*DMS-DDL text ANALYSIS*); análise dos procedimentos do sistema (*PROGRAM ANALYSIS*); exame do conteúdo dos arquivos e bancos de dados (*DATA ANALYSIS*); e integração de esquemas, quando mais de uma fonte de dados for analisada (*SCHEMA INTEGRATION*).

A segunda fase da metodologia consiste na detecção e transformação de estruturas não conceituais, redundâncias, e construções específicas do gerenciador de dados. Esta fase é dividida em dois subprocessos: extração de todos os conceitos semânticos relevantes do esquema lógico (*BASIC CONCEPTUALIZATION*); e reestruturação do esquema conceitual básico, melhorando a qualidade de expressão do esquema conceitual final (*CONCEPTUAL NORMALIZATION*).

Os métodos citados acima, assim como outros presentes na literatura [BAT92], auxiliam na obtenção de esquemas conceituais a partir de dados já existentes. Porém, tais métodos são de propósito geral. As aplicações de SIG manipulam dados com características específicas como, por exemplo, múltiplas representações espaciais e aspectos temporais, as quais dificilmente são tratadas pelos métodos genéricos. Na revisão bibliográfica realizada não foi encontrado nenhum método de engenharia reversa específico para aplicações que manipulam dados geográficos. Apenas a proposta de um ambiente de extração de metadados, chamado MetaSIG [PRE99], foi encontrada.

O MetaSIG é um ambiente de metadados que possui como objetivo a extração automática de esquemas e metadados presentes em aplicações geográficas, independente do modelo utilizado. Para tal, o ambiente utiliza a “arquitetura de quatro camadas” para metamodelagem. A primeira camada, chamada “Dados do Usuário”, refere-se aos dados de aplicações geográficas. A segunda camada, chamada “Esquemas”, contém os esquemas conceituais resultantes da aplicação de um modelo, como, por exemplo, o ER, sobre um determinado domínio de informação. A terceira camada, chamada “Meta-esquemas”, contém os esquemas conceituais cujos domínios de informação referem-se, especificamente, aos modelos de dados, sejam eles conceituais ou físicos. A quarta camada, chamada “Metameta-esquema”, define a linguagem para especificar meta-esquemas. Ou seja, através de um determinado metameta-esquema deve ser possível criar instâncias de meta-esquemas.

A arquitetura do MetaSIG, ilustrada na Figura 1.1, é composta por: fontes dos esquemas de dados e metadados, repositórios, e módulos componentes. Para que seja possível extrair automaticamente metadados das fontes, os conceitos de meta-esquema e metameta-esquema são utilizados. O meta-esquema possibilita a representação, em um alto nível de abstração, dos metadados a serem extraídos de uma determinada fonte. O responsável pela confecção do meta-esquema é denominado “Usuário Integrador”.

São utilizados dois tipos de repositórios no MetaSIG: Banco de Meta-esquemas e Bancos de Metadados. O Banco de Meta-esquemas tem como instâncias os meta-esquemas levantados pelo integrador. Essas instâncias são usadas pelos módulos responsáveis pela extração de metadados e interface com o usuário. Os meta-esquemas dão origem aos Bancos de Metadados. Para cada formato de metadados utilizado em aplicações de SIG, um Banco de Metadados deve ser implementado.

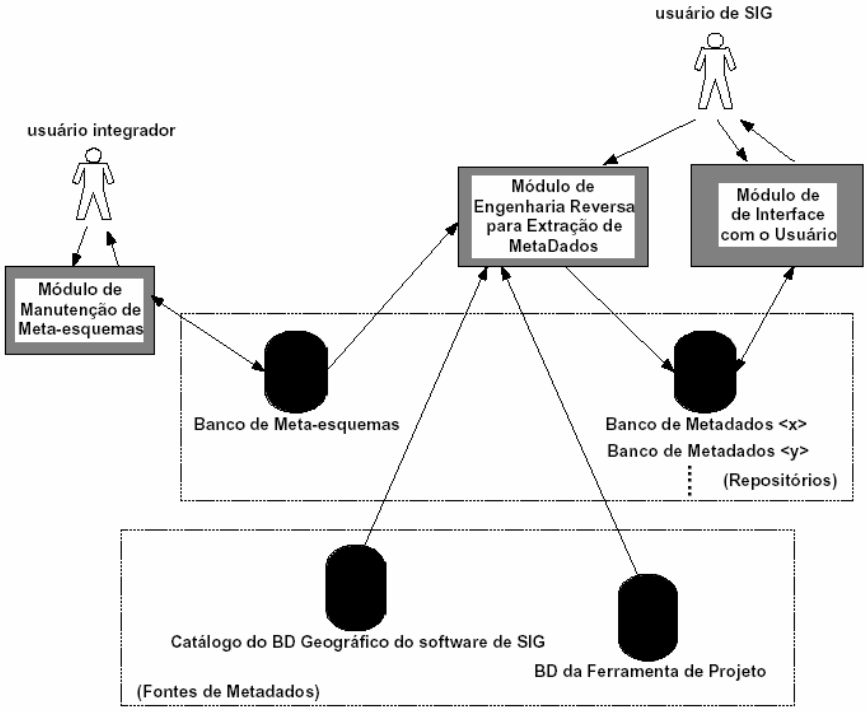

FIGURA 1.1- Arquitetura do Ambiente de Metadados MetaSIG. [PRE99]

Os principais módulos funcionais que compõem o MetaSIG são:

- Módulo de Manutenção de Meta-esquemas: módulo responsável pela entrada e manutenção dos meta-esquemas;
- Módulos de Engenharia Reversa para Extração de Metadados: módulo responsável pela extração automática de metadados;
- Módulo de Interface com o Usuário: módulo responsável pela apresentação dos metadados ao usuário de SIG e pela adição de metadados complementares.

Os objetivos do MetaSIG diferem dos deste trabalho, uma vez que busca a extração de metadados a partir de diversas fontes de dados. Já os procedimentos de engenharia reversa, propostos neste trabalho, buscam extrair as informações necessárias para a criação ou manutenção de um modelo conceitual a partir dos próprios dados geográficos. Para que seja possível a extração dos metadados, através do MetaSIG, o usuário deve ter conhecimento sobre a estrutura dos dados e informá-los ao sistema (Banco de Meta-esquemas). Entretanto, no processo de aquisição de dados de terceiros é comum o usuário não possuir informações sobre a estrutura dos mesmos.

1.3 Objetivos do Trabalho

O objetivo principal da pesquisa realizada foi definir um conjunto de procedimentos de engenharia reversa de bancos de dados espaciais georreferenciado que auxilie na criação e manutenção de modelos conceituais para aplicações de SIG a partir de dados já existentes. Foi dada ênfase à tradução dos objetos armazenados em formatos de transferência de dados geográficos para sua representação em um modelo de dados para aplicações de SIG.

Mais precisamente, o trabalho buscou:

- ✓ definir a arquitetura de um sistema de engenharia reversa para bancos de dados geográficos;
- ✓ estudar a estrutura de formatos de transferência de dados geográficos utilizados por usuários de SIG, identificando formatos que possam ser usados nos processos de engenharia reversa;
- ✓ definir regras que possibilitem o mapeamento dos objetos armazenados no formato de transferência escolhido de dados geográficos para um modelo de dados para aplicações de SIG;
- ✓ verificar a eficácia das regras de mapeamento e da arquitetura do sistema de engenharia reversa através de um estudo de caso.

1.4 Estrutura do Trabalho

O texto do trabalho está estruturado em seis capítulos e dois anexos, além das referências bibliográficas.

O Capítulo 2 propõe uma arquitetura de *software* para o sistema de engenharia reversa. A proposta é composta pela definição da arquitetura, dos módulos, das classes e das características do sistema.

O Capítulo 3 discute alguns dos principais formatos de transferência de dados hoje utilizados em SIG. Introduz o *framework* GeoFrame, o qual deverá servir de base para os esquemas conceituais a serem obtidos e para a definição de padrões de modelagem. Finalmente o Capítulo 3, apresenta o mapeamento das estruturas dos formatos de transferência de dados para o *framework* GeoFrame.

O Capítulo 4 faz o detalhamento do módulo Tradutor, componente da arquitetura proposta no Capítulo 2. Este módulo é responsável pela implementação das regras de mapeamento definidas no Capítulo 3.

O Capítulo 5 descreve um estudo de caso de implementação do sistema proposto no Capítulo 2.

O Capítulo 6 apresenta conclusões e discute possíveis próximos passos do trabalho.

O Anexo 1 apresenta padrões de modelagem que foram extraídos do projeto SIME (Sistema de Informações Georreferenciadas da Região Metropolitana de Belém) e utilizados no estudo de caso apresentado no Capítulo 5. O Anexo 2 apresenta um exemplo de esquema de dados representado em *XML Schema* para um tema chamado Hidrografia.

2 Uma Arquitetura de Sistema de Engenharia Reversa para SIG

Este Capítulo apresenta uma proposta de arquitetura para um mecanismo de engenharia reversa de bancos de dados geográficos capaz de auxiliar na criação e manutenção de modelos conceituais para aplicações de SIG, a partir de dados já existentes. Este mecanismo deve proporcionar um uso mais eficaz de SIG já implementados, além de contribuir para o aumento do intercâmbio de dados entre bancos de dados geográficos.

Os casos de uso da Figura 2.1, em notação UML [BOO2000], definem os serviços fornecidos pelo sistema de engenharia reversa para SIG. O usuário solicita ao sistema que seja realizada a engenharia reversa a partir de uma fonte de dados.

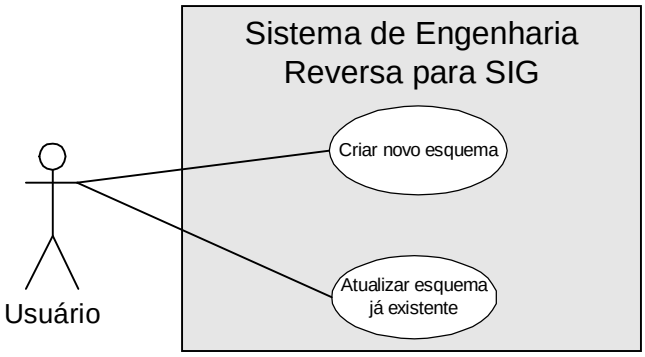

FIGURA 2.1 – Casos de Uso do Sistema de Engenharia Reversa para SIG.

O caso de uso **Criar novo esquema** deve ser capaz de, a partir de uma fonte de dados fornecida pelo usuário, criar um modelo conceitual de um determinado SIG. A Figura 2.2 mostra o diagrama de seqüência deste caso de uso.

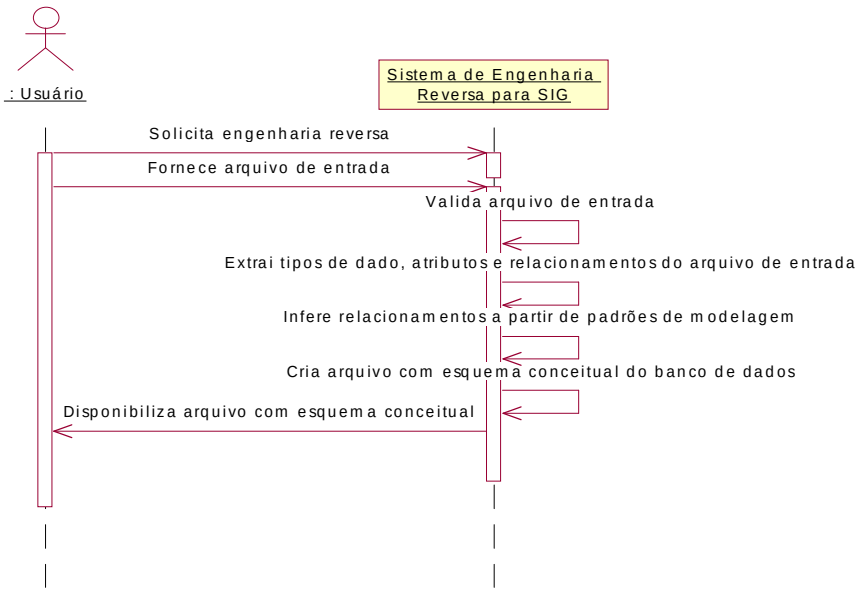

FIGURA 2.2 – Diagrama de seqüência do caso de uso **Criar novo esquema**.

Inicialmente, o usuário solicita que seja realizada a engenharia reversa para uma determinada fonte de dados, fonte esta tratada, aqui, como arquivo de entrada. Este arquivo de entrada deve estar definido em um dos formatos aceitos pelo sistema, sendo que estes podem possuir restrições que limitam o processo de engenharia reversa. Caso o arquivo seja de um formato inadequado, o mesmo será rejeitado impedindo, assim, a continuação do processo de engenharia reversa.

Após a validação e identificação do formato do arquivo de entrada é iniciado, então, o processo de identificação dos tipos de dado existentes no arquivo de entrada. O esquema de dados resultante do processo de engenharia reversa, proposto pelo atual trabalho, está baseado no *framework* GeoFrame apresentado no Capítulo 3. O GeoFrame foi escolhido para servir de base aos modelos que serão criados e mantidos pelo processo de engenharia reversa proposto, neste trabalho, por se tratar de uma representação genérica, independente de aplicação, extensível, e capaz de representar todos requisitos necessários para os modelos de dados inferidos pelo sistema de engenharia reversa aqui proposto. Todos os tipos de dados identificados a partir de uma fonte de dados deverão representar classes da aplicação que, por sua vez, representam especializações das classes do GeoFrame. Para tal, serão aplicadas regras de mapeamento específicas para cada formato de fonte de dados. No Capítulo 3 são estudados três formatos de dados geográficos utilizados por usuários de SIG e definidas suas regras de mapeamento. Caso não seja possível determinar uma classe específica, à qual o tipo de dado deve ser associado, o usuário pode ser chamado a determinar se o mesmo será desprezado ou para indicar uma associação possível. São lidos e analisados todos os tipos de dado, seus atributos e relacionamentos, contidos no arquivo de entrada.

Os relacionamentos só podem ser identificados, caso o formato do arquivo de entrada permita armazer tal informação, o que alguns formatos não fazem. Neste caso, o sistema deve inferir relacionamentos, que não foram identificados no arquivo de entrada, entre os tipos de dado. Para tal, o sistema deve consultar padrões de modelagem para sugerir possíveis relacionamentos para os tipos de dado. O sistema pode apresentar ao usuário um conjunto de padrões que correspondam às classes que fazem parte do esquema do banco de dados geográficos, e o usuário escolherá o que melhor interpreta a realidade de sua aplicação. Caso nenhum dos padrões existentes seja aplicável, o usuário deve ser chamado a definir os relacionamentos que serão adicionados ao esquema da aplicação para integrar os novos tipos.

O conjunto de todas as classes associadas e seus relacionamentos formará o esquema conceitual do banco de dados geográficos. Este será disponibilizado ao usuário através de um arquivo, no formato XML Schema, discutido na Seção 2.1.2.

O caso de uso **Atualizar esquema já existente** deve ser capaz de, a partir de uma fonte de dados fornecida pelo usuário, atualizar o modelo conceitual de um determinado SIG. A Figura 2.3 mostra o diagrama de seqüência deste caso de uso.

O usuário solicita a engenharia reversa, fornece um arquivo de entrada com dados geográficos e um arquivo, no formato XML Schema, com o modelo conceitual a ser atualizado. Após a validação dos arquivos fornecidos, assim como no caso de uso **Criar novo esquema**, o sistema inicia o processo de identificação dos tipos de dados existentes no arquivo de entrada.

Para cada tipo de dado extraído do arquivo de entrada, deve-se determinar se este é novo no esquema conceitual fornecido pelo usuário. Os tipos novos, juntamente com seus atributos e relacionamentos, são utilizados na atualização do esquema.

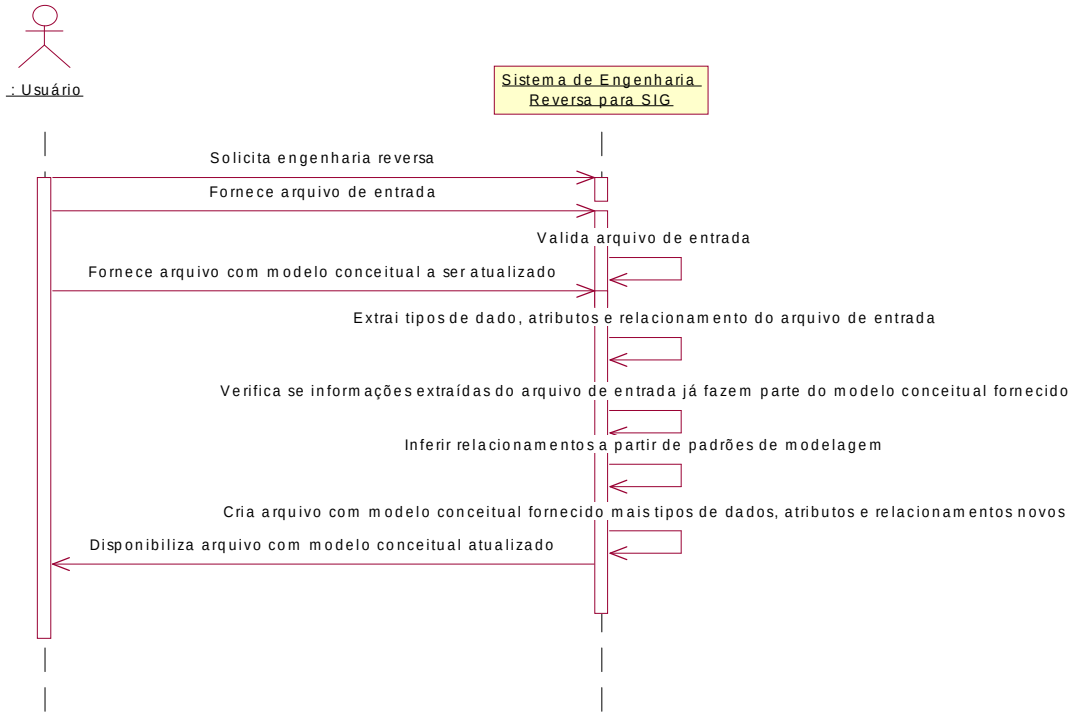

FIGURA 2.3 – Diagrama de seqüência do caso de uso Atualizar esquema já existente.

Para os tipos que já fazem parte do esquema atual, deve ser analisado se apresentam atributos e relacionamentos novos em relação ao esquema atual. Mesmo que uma classe, identificada a partir do arquivo de entrada, já exista no esquema atual do banco de dados, esta pode estar relacionada com novas classes que ainda não fazem parte do esquema ou, ainda, podem existir novos atributos para esta classe.

Os tipos de dado, atributos e relacionamentos são identificados através de seu nome, o que implica em o sistema oferecer suporte ao tratamento de termos sinônimos. O sistema que está sendo proposto define um módulo, chamado Gerenciador de Termos, para resolver os conflitos de nome. Este módulo recebe um nome e um tipo, procura por sinônimos e honônimos, e retorna o nome que deve ser adotado.

Após todos os tipos de dado do arquivo de entrada terem sido analisados, o sistema deve inferir relacionamentos que não foram definidos pelo arquivo de entrada, relacionamentos estes entre os tipos novos, e entre um tipo novo e tipos já existentes no esquema atual. Para tal processo, o sistema deve consultar padrões de modelagem para sugerir possíveis relacionamentos para os tipos de dados, da mesma forma que no caso de uso **Criar novo esquema**.

O conjunto de todos novos tipos de dado, atributos e relacionamentos, extraídos do arquivo de entrada, e os já definidos no esquema conceitual fornecido pelo usuário são integrados e disponibilizados através de um arquivo no formato XML Schema.

2.1 Componentes do Sistema

Para que os casos de uso definidos, acima, possam ser implementados foram definidos os componentes que compõem a arquitetura do sistema de engenharia reversa

para SIG que está sendo proposto. Os componentes e seus relacionamentos são ilustrados através do diagrama da Figura 2.4.

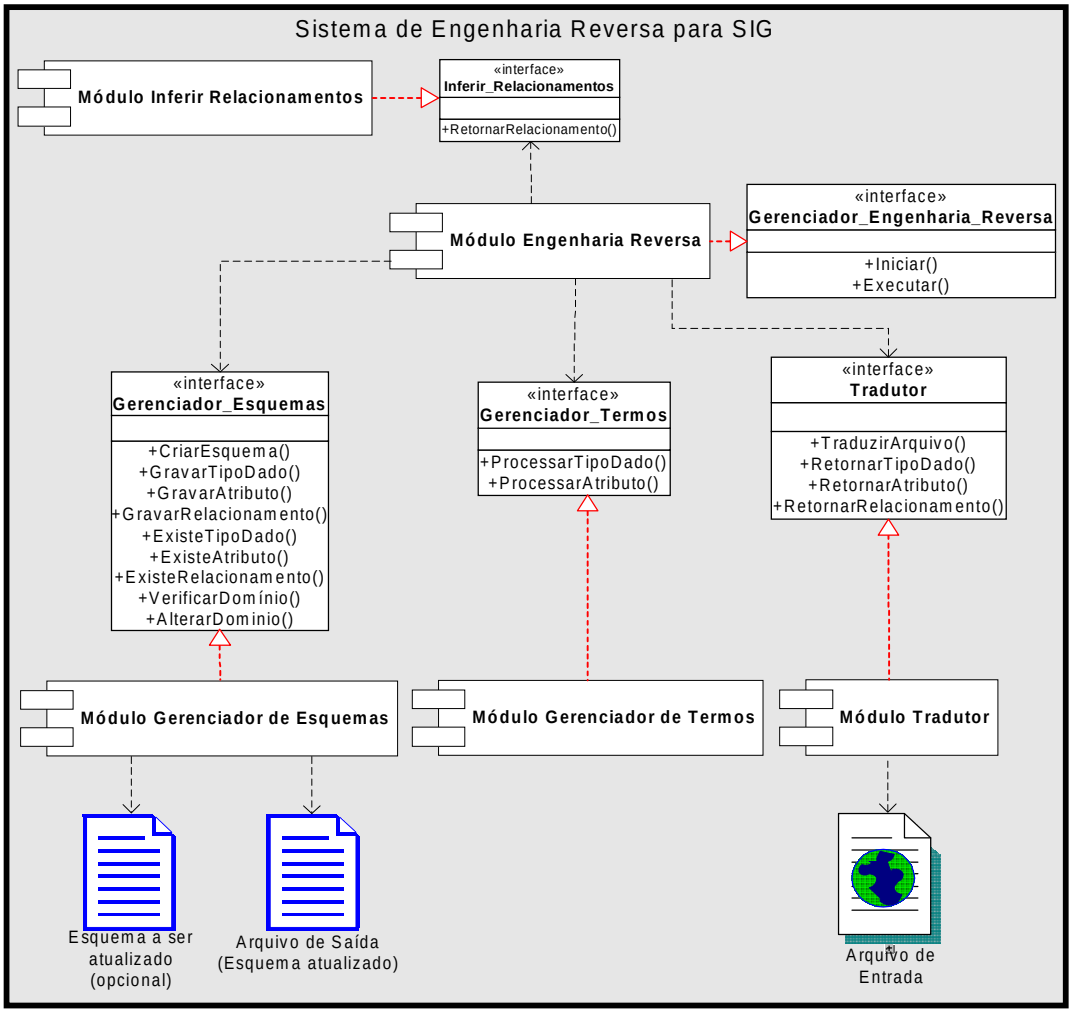

FIGURA 2.4 – Arquitetura do Sistema de Engenharia Reversa para SIG.

O sistema é composto de cinco módulos que disponibilizam seus serviços através de classes de *interface*. Também fazem parte da arquitetura do sistema: um arquivo de entrada contendo dados geográficos; opcionalmente, um arquivo no formato XML Schema contendo um modelo conceitual de dados a ser atualizado; e um arquivo de saída, também no formato XML Schema, contendo o esquema conceitual do banco de dados geográficos gerado a partir dos arquivos de entrada. A seguir serão discutidos cada um dos componentes do sistema.

2.1.1 Arquivo de Entrada

O arquivo de entrada deve ser fornecido pelo usuário ao solicitar a engenharia reversa ao sistema. Este deve conter dados geográficos definidos em um dos formatos aceitos pelo sistema. No Capítulo 3 são discutidos três formatos aceitos pela comunidade de SIG.

2.1.2 Arquivos com Esquema Conceitual

Dois arquivos contendo informações que definem um esquema conceitual de dados são utilizados pelo sistema de engenharia reversa. O primeiro, fornecido pelo usuário, define um esquema conceitual de um banco de dados geográficos que deve ser atualizado com as informações do arquivo de entrada. O segundo arquivo, tratado aqui como arquivo de saída, é o resultado do processo de engenharia reversa. Este último arquivo disponibiliza, ao usuário, o esquema de dados do banco de dados geográfico quando o processo termina.

Estes arquivos são definidos em XML Schema [XML2000] de acordo com as normas da GML 2.0 [OGC2001]. A Geography Markup Language (GML), discutida no Capítulo 3, é uma codificação em XML Schema para transporte e armazenamento de informações geográficas.

O esquema de dados resultante do processo de engenharia reversa proposto, assim como os esquemas fornecidos pelo usuário, para serem atualizados, devem estar baseados no *framework* GeoFrame. Para que um modelo conceitual de dados de um SIG, projetado utilizando o GeoFrame, seja definido em GML, algumas regras de mapeamento devem ser observadas. Estas regras estão definidas em [FOR2002a]. São elas:

- ✓ Regra 1: Cada tema é mapeado para um arquivo em XML Schema;
- ✓ Regra 2: Cada classe é transformada em um tipo;
- ✓ Regra 3: Cada atributo é transformado em um atributo do tipo correspondente;
- ✓ Regra 4: As características geométricas, indicadas pelos estereótipos, são mapeadas para propriedades geométricas adequadas;
- ✓ Regra 5: As características temporais são mapeadas utilizando elementos temporais;
- ✓ Regra 6: Cada relação é descrita, explicitamente, por um tipo;
- ✓ Regra 7: Cada hierarquia de generalização em GeoFrame é mapeada para uma hierarquia de generalização de tipos em XML Schema.

No Anexo 2 é apresentado um esquema XML que descreve o tema hidrografia, modelado com base no GeoFrame. Este tema é apresentado na Figura 2.5. Para codificação deste esquema foram observadas as normas da GML 2.0 e as regras citadas acima.

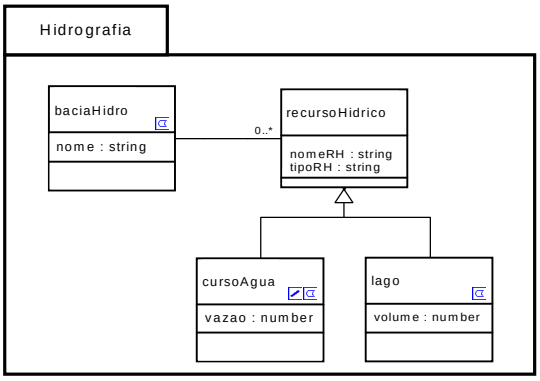

FIGURA 2.5 – Tema Hidrografia modelado em GeoFrame.

2.1.3 Módulo Engenharia Reversa

O módulo Engenharia Reversa é responsável por implementar a *interface* com o usuário e gerenciar o processo de engenharia reversa como um todo. A Figura 2.6 mostra o diagrama de classes deste módulo. A classe Gerenciador_Engenharia_Reversa define as funcionalidades do módulo, enquanto as demais são utilizadas para definir o tipo dos atributos desta classe.

Quando o usuário solicita a engenharia reversa para uma fonte de dados, um objeto da classe Gerenciador_Engenharia_Reversa é instânciado e o método Iniciar() é invocado.

A classe TipoDado define a estrutura dos tipos de dado que são estraídos do arquivo de entrada. Possui dois atributos: Nome descreve o nome do tipo de dado. ClasseGeoFrame informa a qual das classes do *framework* este tipo de dado está associado.

A classe Atributo define a estrutura de um atributo, de um determinado tipo de dado, que tenha sido extraído do arquivo de entrada. Possui três atributos: Nome indica o nome do atributo. Tipo indica o domínio do atributo. Tamanho informa o tamanho do atributo.

A classe Relacionamento define a estrutura de um relacionamento entre dois tipos de dado extraídos do arquivo de entrada. Possui sete atributos: Tipo indica o tipo de associação entre os tipos de dados. Pode assumir um dos seguintes valores: associação, agregação e generalização. TipoDado1 e TipoDado2 informam o nome dos tipos de dado que estão sendo associados. CardMin1, CardMax1, CardMin2, e CardMax2 definem as cardinalidades mínimas e máximas do relacionamento.

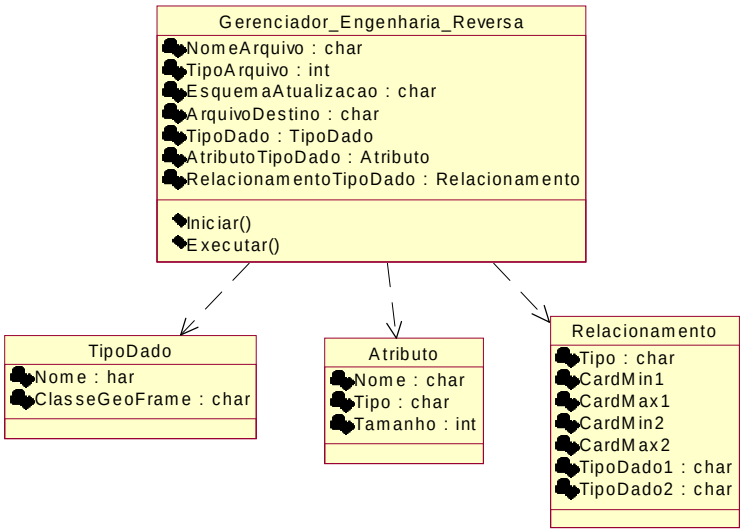

FIGURA 2.6 – Diagrama de classes do módulo Engenharia Reversa.

A classe Gerenciador_Engenharia_Reversa é responsável pelo controle de todo o processo de engenharia reversa. Possui seis atributos: NomeArquivo armazena o nome do arquivo de entrada que será processado. TipoArquivo informa o formato do arquivo de entrada. EsquemaAtualizacao indica o nome do arquivo que contém o esquema de dados que deve ser atualizado, este atributo é opcional. TipoDado recebe a instância de um tipo de dado. AtributoTipoDado corresponde à

instância de um determinado atributo. Por fim, RelacionamentoTipoDado corresponde a cada relacionamento do tipo de dado. Os três últimos atributos são atualizados toda vez que um novo tipo de dado, atributo, ou relacionamento é extraído do arquivo de entrada. A classe possui, ainda, os métodos Iniciar() e Executar().

O método Iniciar() é responsável pela interação inicial com o usuário. A Figura 2.7 mostra o diagrama de atividades deste método.

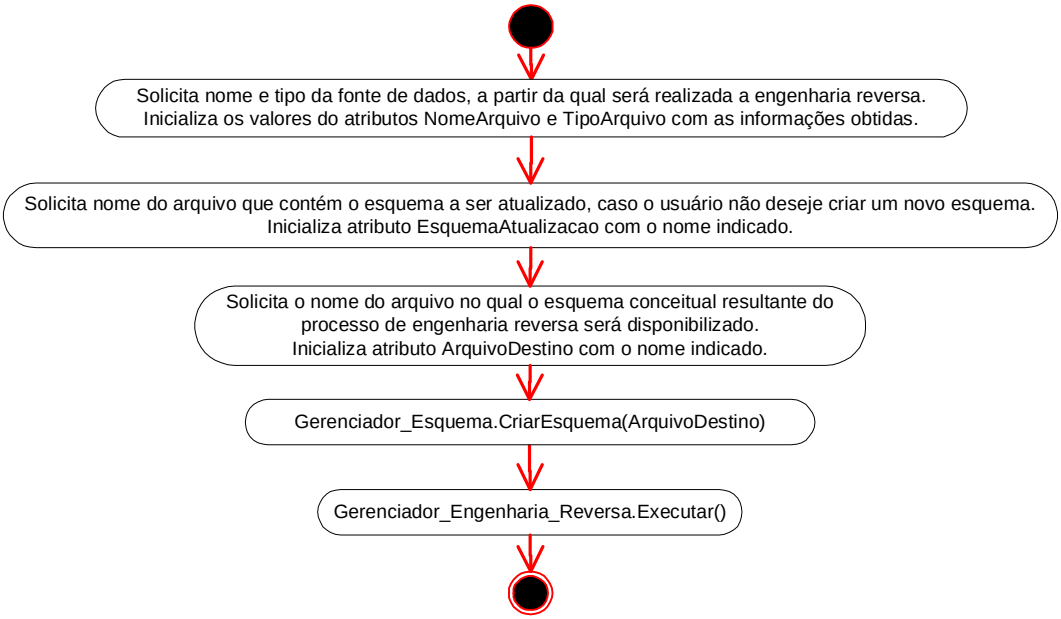

FIGURA 2.7 – Diagrama de atividades do método Iniciar da classe Gerenciador_Engenharia_Reversa.

Inicialmente é solicitado ao usuário o nome dos arquivos de entrada, saída e, caso o usuário deseje, um arquivo com o esquema para ser atualizado. São atualizados os atributos da classe referentes às informações acima. É invocado, então, o método CriarEsquema() da classe Gerenciador_Esquema, responsável pela criação do arquivo de saída. Em seguida o método Executar() da classe será invocado iniciando, assim, o processo de engenharia reversa.

O método Executar() gerencia todo o processo de engenharia reversa a partir de um arquivo de entrada. A Figura 2.8 mostra o diagrama de atividades deste método. Ele chama os métodos responsáveis por cada uma das funções do processo, executando as seguintes atividades:

a) Inicia a tradução do arquivo de entrada, disparando o método TraduzirArquivo() da classe Tradutor. Passa, como parâmetros, o nome do arquivo de entrada, e seu tipo. Caso o arquivo não seja compatível com o formato que foi informado, receberá um erro como retorno. Neste caso, informará ao usuário que o arquivo é inválido e que a engenharia reversa não pode ser feita;

b) Solicita ao módulo Tradutor que retorne um tipo de dado, extraído do arquivo de entrada. Isto é feito invocando o método RetornarTipoDado() da classe Tradutor. Para cada tipo de dado obtido, serão realizadas as atividades descritas a partir da atividade (c).

b.1) Quando não existirem mais tipos de dado a serem processados, o método RetornarRelacionamento() da classe Inferir_Relacionamentos será invocado. Este método tentará inferir

relacionamentos entre os novos tipos e, se um esquema estiver sendo atualizado, os tipos já existentes. Cada relacionamento retornado será agregado ao esquema de dados final;

b.2) Quanto não houver mais relacionamentos, a nova versão do esquema de dados pode ser disponibilizada ao usuário;

c) Chama o método ProcessarTipoDado() da classe Gerenciador_Termos, passando, como parâmetro, o nome do tipo de dado que está sendo processado. Este método irá procurar por sinônimos e honônimos e indicará o nome que deve ser adotado para o tipo de dado, resolvendo, assim, eventuais conflitos de nomes que possam existir;

d) Procura no esquema, sendo atualizado, por um tipo de dado com o mesmo nome. Para tal, chama o método ExisteClasse() da classe Gerenciador_Esquemas. Passa como parâmetros o tipo de dado, e recebe, como resposta, se este já existe no esquema. Se não existir, chama o método GravarClasse() da classe Gerenciador_Esquemas. Este irá agregar o novo tipo de dado ao esquema do arquivo de saída;

e) Obtém, chamando o método RetornarAtributo() da classe Tradutor, todos os atributos do tipo de dado que está sendo processado. Para cada um dos atributos, executa as seguintes atividades:

e.1) Resolve conflitos de nomes, através do método ProcessarAtributo() da classe Gerenciador_Termos. Passa como parâmetro o nome do atributo e recebe, em retorno, o nome que deve ser adotado;

e.2) Verifica se o atributo já existe no esquema, para o tipo de dado que está sendo processado. Chama o método ExisteAtributo() da classe Gerenciador_Termos, passando como parâmetro o tipo de dado e o atributo. Recebe, como resposta, se o atributo existe. Se o atributo não existir vai para a atividade (e.5);

e.3) O atributo já existe no esquema da aplicação. Neste caso, é necessário verificar se o domínio, do atributo que está sendo processado e do atributo que já existe no esquema, são iguais. Para isso, chama o método VerificarDominio() da classe Gerenciador_Esquemas, passando como parâmetros o tipo de dado e o atributo. Caso os domínios sejam iguais o atributo é ignorado;

e.4) Quando os domínios não são iguais, é solicitado ao usuário que indique se deseja ignorar o atributo, agregá-lo ao esquema com outro nome ou, ainda, mudar o domínio atual pelo novo. Caso queira agregá-lo, um novo nome para o atributo será solicitado ao usuário;

e.5) Caso o novo atributo não tenha sido ignorado em (e.4), este será agregado ao esquema através do método GravarAtributo() da classe Gerenciador_Esquemas, ou o domínio do atributo será modificado através do método AlterarDominio();

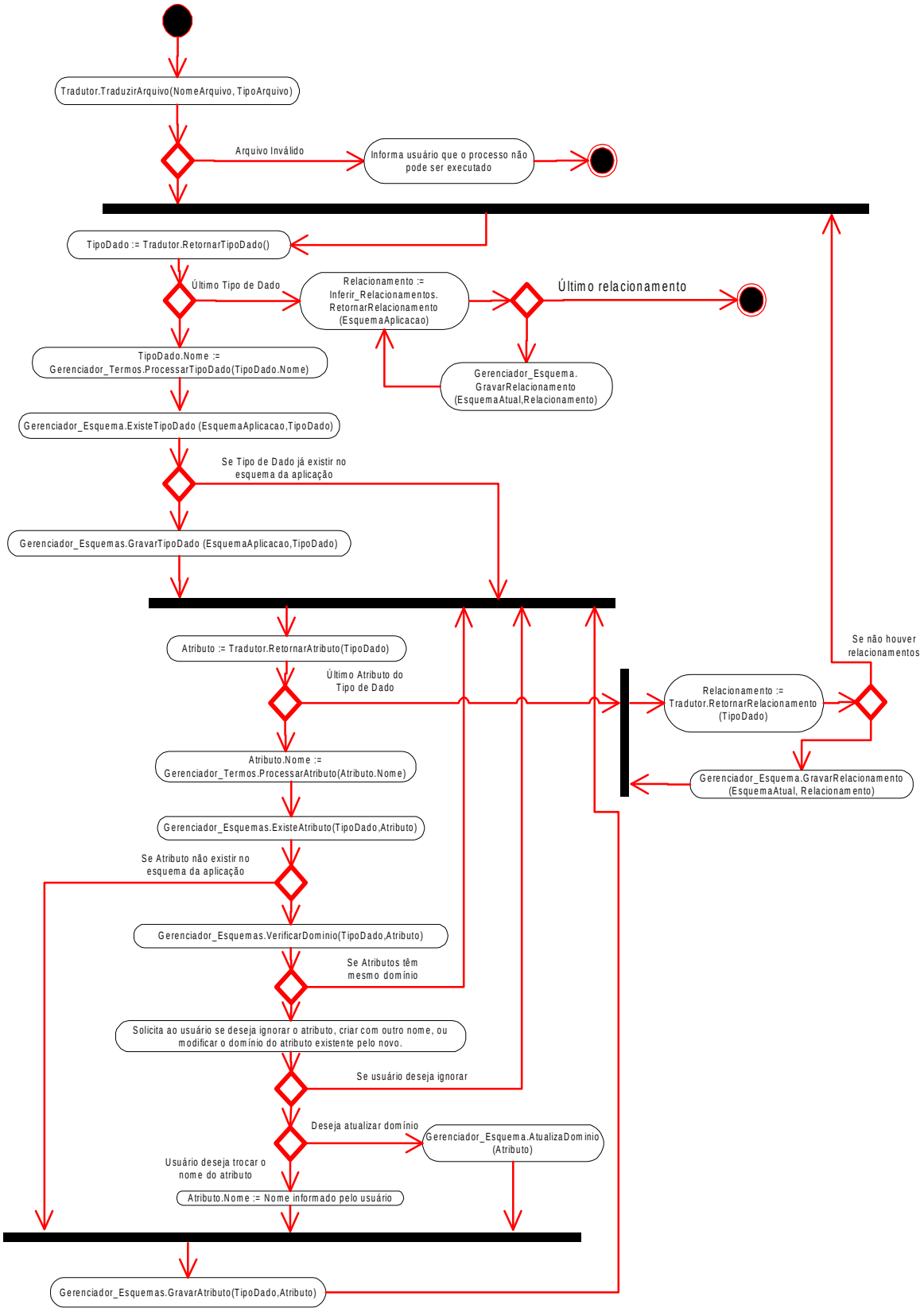

FIGURA 2.8 – Diagrama de Atividades do método Executar da classe Gerenciador_Engenharia_Reversa.

f) Quando todos os atributos do tipo de dado tiverem sido processados, busca seus relacionamentos. Esta atividade só será possível caso o formato do arquivo de entrada disponibilize tal informação. Se não houver relacionamentos para o tipo de dado, ou o formato não disponibilizar este tipo de informação, reinicia-se este algoritmo a partir da atividade (b) acima. Os relacionamentos são obtidos, chamando-se o método RetornarRelacionamento() da classe Tradutor. Passa-se, como parâmetro, o tipo de dado e recebe-se, como retorno, um relacionamento;

g) Para cada relacionamento identificado, agrega-o ao esquema, chamando o método GravarRelacionamento() da classe Gerenciador_Esquemas. Caso o relacionamento seja com um tipo de dado que ainda não faça parte do esquema, um erro será retornado. Neste caso, ou, ainda, se o relacionamento já está definido no esquema, o relacionamento será ignorado.

2.1.4 Módulo Tradutor

O módulo Tradutor, por solicitação do módulo Engenharia Reversa, traduz o conteúdo de um arquivo de formato aceitável para um conjunto de tipos de dado e atributos. Dependendo do formato de entrada, um segundo conjunto pode ser fornecido, este com os relacionamentos entre os novos tipos de dado. Este módulo será detalhado no Capítulo 4.

A classe de *interface* que disponibiliza os serviços desse módulo chama-se Tradutor, ilustrada na Figura 2.9.

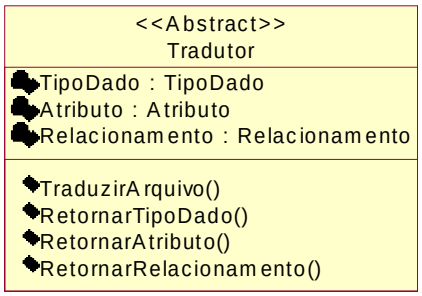

FIGURA 2.9 – Classe Tradutor.

Tradutor é a classe abstrata responsável pela extração dos tipos de dados do arquivo de entrada, seus atributos e relacionamentos. As especializações dessa classe são responsáveis pela tradução de cada um dos formatos de arquivo suportados pelo sistema. Novas especializações para a tradução de outros formatos de arquivo podem ser incorporadas ao sistema a qualquer tempo.

Três são os atributos dessa classe. TipoDado tem como domínio um objeto da classe TipoDado e armazena, a cada momento, o último tipo de dado extraído do arquivo de entrada. Atributo mantém o último atributo extraído do arquivo de entrada, seu domínio é um objeto da classe Atributo. Relacionamento representa, a cada momento, o último relacionamento extraído do arquivo de entrada, é um objeto da classe Relacionamento. Os domínios dos atributos descritos, acima, são classes definidas no módulo Gerenciador Engenharia Reversa.

A classe Tradutor possui quatro métodos a serem implementados por suas especializações de acordo com a necessidade de cada formato de entrada aceito. No

Capítulo 4 as especializações desta classe são definidas e os métodos detalhados. Os métodos são:

- ✓ TraduzirArquivo(): recebe o nome de um arquivo, contendo dados geográficos, e seu formato. Valida o conteúdo do arquivo, de acordo com seu formato, retornando um erro se necessário. Inicializa os atributos da classe para que seja possível a extração dos tipos de dados;
- ✓ RetornarTipoDado(): retorna um novo tipo de dado extraído do arquivo de entrada. Quando não houver mais tipos de dado, retorna uma mensagem de fim;
- ✓ RetornarAtributo(): recebe um tipo de dado e retorna cada um de seus atributos. Quando não houver mais atributos para o tipo de dado, retorna uma mensagem de fim;
- ✓ RetornarRelacionamento(): recebe o tipo de dado e retorna cada um de seus relacionamentos. Isto só é possível se o formato do arquivo permitir a representação de tal informação. Quando não houver mais relacionamentos, retorna uma mensagem de fim.

2.1.5 Módulo Gerenciador de Termos

O módulo Gerenciador de Termos deve ser capaz de resolver conflitos de nomes que ocorram entre tipos de dado e atributos de tipos de dado, que estejam sendo agregados ao esquema de dados, e os tipos de dados e atributos que já fazem parte do esquema, respectivamente. A classe Gerenciador_Termos, ilustrada na Figura 2.10, é a *interface* deste módulo. Para esta classe foram definidos dois métodos: ProcessarTipoDado() e ProcessarAtributo(). Estes recebem um nome como parâmetro, procuram por sinônimos e honônimos, e retornam um nome padrão que deve ser adotado. Objetos como Thesaurus podem ser usados para a resolução dos conflitos de nomes.

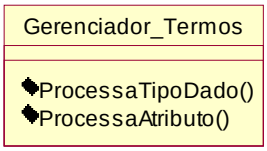

FIGURA 2.10 – Classe Gerenciador_Termos.

Os conflitos de nome existem já que as fontes de dados utilizadas na atualização de esquemas podem ser provenientes de diferentes fornecedores. As feições do mundo real, representadas pelos tipos de dados, podem ser interpretadas de formas diferentes pelos cartógrafos responsáveis pela compilação dos dados de cada fornecedor. É relevante, também, o fato de que nomes de tipos e atributos possam estar armazenados em idiomas distintos.

É necessário evitar que classes que representem a mesma realidade geográfica sejam diferenciadas apenas por seu nome. Similarmente, deve-se evitar que classes com o mesmo nome, mas representando diferentes realidades geográficas sejam consideradas iguais. O SDTS [ANS97], consciente deste problema, propõe, na parte 2 de sua especificação, uma lista de tipos de dado espaciais com respectivos nomes padrão e demais definições que devem ser utilizados na transferência de dados por este padrão.

2.1.6 Módulo Inferir Relacionamentos

Este módulo é responsável por propor relacionamentos, por solicitação da classe Engenharia_Reversa, entre tipos novos, e entre um tipo novo e tipos já existentes no esquema que está sendo atualizado, a partir de pesquisa a um catálogo de padrões. A definição de um catálogo de padrões, sua manutenção, a definição de regras para escolha dos padrões, formas de recuperação de informações, e outras funcionalidades não são objetivos deste trabalho e, portanto, devem ser propostos e descritos em futuros trabalhos, complementares a este.

A classe Inferir_Relacionamentos, ilustrada na Figura 2.11, define a interface do módulo. Para esta classe está definido o método RetornarRelacionamento(). Este recebe, como parâmetro, um esquema e deve retornar um relacionamento entre dois tipos que fazem parte do esquema indicado. O relacionamento retornado é agregado ao esquema de dados. Este método será chamado enquanto houver relacionamentos a serem sugeridos.

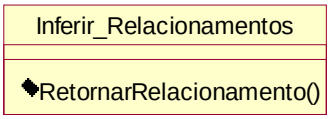

FIGURA 2.11 – Classe Inferir_Relacionamentos.

2.1.7 Módulo Gerenciador de Esquemas

O módulo Gerenciador de Esquemas é responsável pela manipulação de esquemas conceituais, baseados no *framework* GeoFrame, que estejam sendo gerados, ou atualizados pelo sistema de engenharia reversa. Deve ser capaz de interpretar o arquivo, no formato XML Schema, que contém o esquema de dados a ser atualizado. Além disso, deve gerar um arquivo de saída, no formato XML Schema, seguindo as regras definidas na Seção 2.1.2, contendo o esquema de dados resultante do processo de engenharia reversa.

A classe Gerenciador_Esquemas é a *interface* deste módulo com os demais módulos do sistema. Ela é ilustrada na Figura 2.12. A classe tem nove métodos que definem suas funcionalidades. A seguir, descreve-se cada um deles.

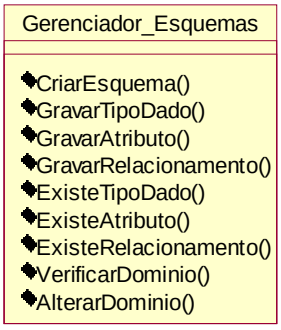

FIGURA 2.12 – Classe Gerenciador_Esquemas.

O método CriarEsquema() recebe, como parâmetro, o nome do arquivo de saída do processo de engenharia reversa. Cria um arquivo com o nome informado e gera a estrutura básica de um XML Schema para o esquema de dados que será definido, baseado em GeoFrame, através do processo de engenharia reversa. Os métodos

GravarTipoDado(), GravarAtributo(), e GravarRelacionamento() agregam informações à estrutura criada por este método.

Caso um esquema tenha sido indicado pelo usuário para ser atualizado, os tipos que definem as classes que compõem este esquema serão extraídos e agregados ao XML Schema que está sendo criado. Cada tipo de dado, extraído deste arquivo, deve ser processado pelo método ProcessarTipoDado() do módulo Gerenciador de Termos. O nome padrão retornado pelo método deve ser utilizado na agregação do tipo de dado ao esquema que está sendo criado.

O esquema XML, apresentado abaixo, é um exemplo da estrutura criada pelo método. Neste exemplo o arquivo gerado recebe o nome "hidrografia.xsd" e o esquema definindo é um tema chamado "Hidrografia".

No início estão as definições que compõem o cabeçalho de um arquivo XML Schema. Em seguida, aparecem as declarações globais dos elementos tema, temaMember e _temaFeature, seguindo a regra número apresentada na subseção 2.1.2. Esses elementos são utilizados para indicar que as diversas classes a serem definidas posteriormente pertencem ao mesmo tema.

No final, o tipo idPropertyType é definido. Este é utilizado para definir os identificadores de objetos que são utilizados nas associações de classes. A função deste tipo está definida na regra número seis em [FOR2002a].

```
<!-- ============================================================
      Cabeçalho XML Schema
 ============================================================ -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: hidrografia.xsd  -->
<schema targetNamespace="hidrografia" mlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml=http://www.opengis.net/gml xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:hid="hidrografia" elementFormDefault="qualified" version="2.03">
<annotation>
        <appinfo>hidrografia.xsd v2.03 2001-02</appinfo>
        <documentation xml:lang="en">
            GML schema for the hidrography example
        </documentation>
</annotation>
<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/namespaces/gml/core/"
schemaLocation="http://www.opengis.net/namespaces/gml/core/feature.xsd"/>

<!-- ============================================================
      Declarações globais de elementos
 ============================================================ -->
<element name="tema" type="hid:temaType"    substitutionGroup="gml:_FeatureCollection"/>
<element name="temaMember" type="hid:temaMemberType"
    substitutionGroup="gml:featureMember"/>
<element name="_temaFeature" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="gml:_Feature"/>

<!-- ============================================================
      Definições de tipos
 ============================================================ -->
<complexType name="temaType">
        <complexContent>
                <extension base="gml:AbstractFeatureCollectionType">
                        <sequence>
                                <element name="name" type="string"/>
                                <element name="dateCreated" type="date"/>
                        </sequence>
```

```
                </extension>
        </complexContent>
</complexType>

<complexType name="temaMemberType">
        <complexContent>
                <restriction base="gml:FeatureAssociationType">
                        <sequence minOccurs="0">
                                <element ref="hid:_temaFeature"/>
                        </sequence>
                        <attributeGroup ref="gml:AssociationAttributeGroup"/>
                </restriction>
        </complexContent>
</complexType>

<!— ========================================================================
    Neste local estão definidas das classes que compõem o tema Hidrografia
=========================================================================-->

<!— ========================================================================
    Fim da definição das classes que compõem o tema Hidrografia
=========================================================================-->

<!-- Id Property -->
<complexType name="idPropertyType">
        <sequence minOccurs="0">
                <element name="objId" type="id"/>
        </sequence>
        <attributeGroup ref="xlink:simpleLink"/>
</complexType>

</schema>
```

FIGURA 2.13 – Exemplo de XML Schema para tema Hidrografia.

O método GravarTipoDado() recebe, como parâmetro, um objeto da classe TipoDado. Agrega a definição do tipo de dado, passado como parâmetro, ao XML Schema criado no método CriarEsquema(). São utilizadas as regras definidas em [FOR2002a] para definição do tipo de dado. Um novo tipo de elemento é criado para o tipo de dado, de acordo com a regra dois. O tipo de geometria é definida através de um propriedade geográfica adequada, de acordo com a regra quatro definida na subseção 2.1.2. A Figura 2.14 ilustra um exemplo de definição para um tipo de dado chamado “baciaHidro”. No exemplo, a definição da propriedade extentOf indica a propriedade geométrica do tipo, o qual está associado à classe Polígono do *framework* GeoFrame. Na Seção 3.3.2 estão definidas as regras para mapeamento das propriedades geométricas em GML para classes do GeoFrame.

```
<complexType name="baciaHidroType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <sequence>
                                <element ref="gml:extentOf"/>
                        </sequence>
                </extension>
        </complexContent>
</complexType>
```

FIGURA 2.14 – Definição de um tipo de dado em XML Schema.

O método GravarAtributo() recebe, como parâmetros, um objeto da classe TipoDado e um objeto da classe Atributo. Agrega o atributo ao tipo de dado passado como parâmetro. O método busca a definição do tipo de dado no XML Schema, definido pelo método GravarTipoDado(), e agrega a definição do atributo passado

como parâmetro. Na Figura 2.15 está ilustrado um exemplo de definição de um atributo chamado “nome” ao tipo de dado “baciaHidro”.

```
<complexType name="baciaHidroType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <sequence>
                                <element ref="gml:extentOf"/>
                        </sequence>
                        <atribute name=”nome” type=”string”/>
                </extension>
        </complexContent>
</complexType>
```

FIGURA 2.15 – Definição de um atributo em XML Schema.

O método GravarRelacionamento() recebe, como parâmetros, um objeto da classe TipoDado e um objeto da classe Relacionamento. Inicia invocando o método ExisteRelacionamento(), este irá retornar “verdadeiro” caso o relacionamento, passado como parâmetro, já existir no esquema atual. Neste caso, nenhuma atualização será feita. Caso o relacionamento não exista o mesmo será atualizado no esquema atual conforme a seguir.

Se o tipo do relacionamento, passado como parâmetro, for uma associação, um novo tipo deve ser criado no XML Schema, seguindo a regra número seis apresentada na subseção 2.1.3. Na Figura 2.16 está ilustrado um exemplo de associação entre os tipos de dado “baciaHidro” e “recursoHidrico”.

```
<complexType name="bacia_recursoType">
   <annotation>
       <appinfo>baciaHidroId--baciaHidroType.fid;recursoHidricoIdId--
cursoAguaType.fid,lagotype.fid</appinfo>
   </annotation>
   <complexContent>
        <restriction base="gml:FeatureAssociationType">
            <sequence>
               <element ref="hid:baciaHidroId"/>
               <element ref="hid:recursoHidricoId" minOccurs="0" maxOccurs="unbounded"/>
             </sequence>
             <attributeGroup ref="gml:AssociationAttributeGroup"/>
        </restriction>
    </complexContent>
</complexType>
```

FIGURA 2.16 – Definição de um associação entre tipos de dado em XML Schema.

Se o relacionamento for do tipo generalização, uma hierarquia de generalização de tipos em XML Schema é criada na definição do tipo de dado passado como parâmetro, seguindo a regra número sete apresentada na subseção 2.1.2. Na Figura 2.17 está ilustrado um exemplo de generalização entre os tipos “recursoHidrico” e “Lago”.

```
<complexType name="lagoType">
        <complexContent>
                <extension base="hid:recursoHidricoType">
                  <sequence>
                        <element ref="gml:extentOf"/>
                  </sequence>
                  <attribute name="volume" type="number" />
                </extension>
        </complexContent>
</complexType>
```

FIGURA 2.17 – Definição de uma generalização entre tipos de dado em XML Schema.

O método ExisteTipoDado() recebe, como parâmetro, um objeto da classe TipoDado. Ele analisa todas as definições de tipos de dado no XML Schema que está sendo gerado. Se encontrar uma definição para um tipo de dado com o mesmo nome do que foi passado como parâmetro retorna os valor “verdadeiro”. Caso não seja encontrada um tipo com mesmo nome, retorna o valor “falso”;

O método ExisteAtributo() recebe, como parâmetros, um objeto da classe TipoDado e um da classe Atributo. Busca pela definição, no XML Schema, do tipo de dado. Analisa a existência da definição de um atributo, dentro da definição do tipo, que tenha o mesmo nome do que o do atributo passado como parâmetro. Caso os atributos tenham o mesmo nome, retorna o valor “verdadeiro”. Caso o atributo não exista, retona o valor “falso”.

O método ExisteRelacionamento() recebe, como parâmetro, um objeto da classe Relacionamento. Se o tipo do relacionamento for uma associação, busca por uma definição de tipo no XML Schema que defina a associação entre os tipos de dado definidos pelo relacionamento passado como parâmetro. Se a definição existir, retorna o valor “verdadeiro”. Caso contrário, o valor “falso” será retornado.

Se o tipo do relacionamento for uma generalização, busca pela definição do tipo de dado definido pelo relacionamento, analisando se existe uma definição de hierarquia de generalização. Se encontrar a definição, retorna o valor “verdadeiro”. Caso contrário o valor “falso” será retornado.

O método VerificarDominio() recebe, como parâmetros, um objeto da classe TipoDado e um da classe Atributo. Busca pela definição do atributo no tipo de dado passado como parâmetro. Retorna o valor “verdadeiro” se o domínio for igual e “falso” caso não seja.

O método AlterarDominio(), recebe como parâmetros, um objeto da classe TipoDado e um da classe Atributo. Busca pela definição do atributo no tipo de dado. Modifica o tipo do atributo de acordo com os valores definidos pelo parâmetro.

3 Mapeamento entre Formatos de Transferência e Modelo de Dados

Além de definir a arquitetura de um sistema de engenharia reversa para Bancos de dados geográficos, este trabalho buscou estudar a estrutura de formatos de transferência de dados geográficos com o objetivo de definir regras que possibilitem o mapeamento dos objetos, armazenados em um determinado formato de transferência de dados geográficos, para um modelo de dados para aplicações de SIG.

Neste Capítulo são discutidos formatos de transferência de dados geográficos. Além disso, revisa-se um modelo de dados para aplicações de SIG e definem-se regras de mapeamento dos formatos de dados geográficos estudados para o modelo de dados escolhido. Esses elementos são utilizados pelo módulo Tradutor, o qual é detalhado no próximo capítulo.

3.1 Formatos de Transferência de Dados Geográficos

Para que seja possível a transferência de informações geográficas é necessário que sejam estabelecidos formatos de exportação e importação de dados. Normalmente, tais formatos incluem linguagens de especificação, transferência de dados, geocodificação e documentação de metadados e formatos [CAM96].

A preocupação pela definição de um padrão de metadados e um formato para transferência de dados é internacional. Todos os países que utilizam SIG estão envolvidos na definição de um padrão a ser seguido, muitos deles desenvolveram seus próprios padrões de transferência. A ISO (*International Standard Organization*) criou em 1994 o Comitê Técnico ISO 211 voltado para informações geográficas e geomáticas. A norma ISO 8211 (BS 6690) especifica uma descrição de arquivos para a troca de informação, descrição esta formada por um dicionário seguido dos dados propriamente ditos. O OGC (*Open Gis Consortium*), criado em 1994, é uma associação comercial que está criando novas padronizações técnicas e comercias para garantir interoperabilidade em geoprocessamento. O OGC tem a especificação de um *framework* para *software* que acesse dados geográficos e apresente recursos de geoprocessamento distribuído chamada Especificação OpenGIS [OGC99]. Esta especificação pretende prover aos desenvolvedores de *software*, do mundo todo, um modelo de interface comum e detalhado, para que desenvolvam *software* capazes de interoperar com outros no formato OpenGIS. A Especificação OpenGIS está próxima a se tornar um padrão formal de tecnologia de informação [BOR2000]. Porém, nenhuma destas iniciativas é ainda um padrão de fato.

No Brasil, não existe um padrão estabelecido para intercâmbio de dados geográficos. A definição de um padrão cartográfico de intercâmbio que preserve toda a riqueza da informação geográfica vem sendo debatida de forma inicial na Comissão de Cartografia (ComCar), órgão do Ministério do Planejamento, encarregada da normatização das atividades cartográficas no Brasil, e que conta com representantes de instituições como IBGE (Instituto Brasileiro de Geografia e Estatística), DSG (Diretoria do Serviço Geográfico) e INPE (Instituto Nacional de Pesquisas Espaciais) [BAR99].

A maioria das instituições que manipulam informações geográficas armazena seus dados em arquivos, no formato oferecido pelo *software* que é utilizado para

manipulá-los. Existem diversas ferramentas comerciais destinadas à manipulação e análise de dados geográficos. Cada uma destas ferramentas trabalha com diversos formatos de importação de dados e, normalmente, possui um único formato interno de arquivo. Alguns destes formatos fornecem apenas informações sobre a geometria dos objetos e alguns poucos atributos descritivos, outros são formatos completos com linguagem de especificação e metadados associados.

Para que seja possível realizar a engenharia reversa dos dados de um SIG, com o objetivo de atualizar ou criar um modelo conceitual de seu banco de dados geográficos, é necessário conhecer o formato de armazenamento dos dados. Dentro do contexto deste trabalho, foram escolhidos três formatos de dados geográficos que serão utilizados para definir um conjunto mínimo de tipos de dados necessários à engenharia reversa de informações geográficas. São eles: Shapefile [ESR98], SAIF/CSN [BRI95] e GML [OGC2001].

O Shapefile, desenvolvido pela ESRI (*Environmental Systems Research Institute*), foi escolhido por ser muito popular e entendido por quase todos os *software* de SIG do mercado, embora não disponibilize muitas informações para a engenharia reversa.

A GML (*Geography Markup Language*) foi o segundo formato escolhido. Um dos motivos para a escolha desta linguagem foi por ela ter sido desenvolvida pelo consórcio OpenGIS. O OpenGIS pode se tornar um padrão de fato em breve, tornando a GML uma possível linguagem padrão para transferência de informações geográficas. Outro motivo que justifica a escolha desta linguagem é o fato de ela ser uma aplicação em XML (*eXtensible Markup Language*), a qual vem se tornando um padrão de transferência de informações na WEB. O fato de ser um padrão na WEB pode se tornar um fator determinante na escolha do formato em que as informações geográficas serão armazenadas no futuro.

O último formato escolhido é o SAIF (*Spatial Archive and Interchange Format*), sua escolha deve-se ao fato de ele possuir um modelo de dados semelhante ao GeoFrame, o qual será a base para o esquema de dados resultante da engenharia reversa proposta por esse trabalho, e por ser o formato base da ferramenta FME (*Feature Manipulation Engine*). Esta ferramenta permite a conversão entre mais de cinqüenta formatos de dados espaciais e é muito útil quando se possuí dados armazenados em diferentes formatos e necessita-se integrá-los. Todos os formatos que a FME aceita podem ser convertidos para o SAIF. [SAF2001]

3.1.1 Shapefile

Este formato de arquivo armazena geometria e atributos descritivos para feições espaciais de um determinado conjunto de dados. O formato Shapefile consiste de um arquivo principal (.shp), um arquivo de índice (.shx) e uma tabela de dados no formato dBASE (.dbf). O arquivo principal é composto por registros de tamanho variável, cada qual descrevendo a geometria de uma feição através de uma lista de vértices. No arquivo de índices, cada registro contém um *offset* de um registro correspondente no arquivo principal. A tabela de dados contém os atributos descritivos das feições. Cada linha da tabela de dados tem um relacionamento um-para-um com um registro do arquivo principal.

3.1.1.1 Arquivo Principal

Este arquivo contém um registro de cabeçalho de tamanho fixo seguido por outros registros de tamanho variável. Cada registro de tamanho variável é constituído por um cabeçalho de tamanho fixo e registros de conteúdo com tamanho variável. A Figura 3.1 ilustra a organização deste arquivo.

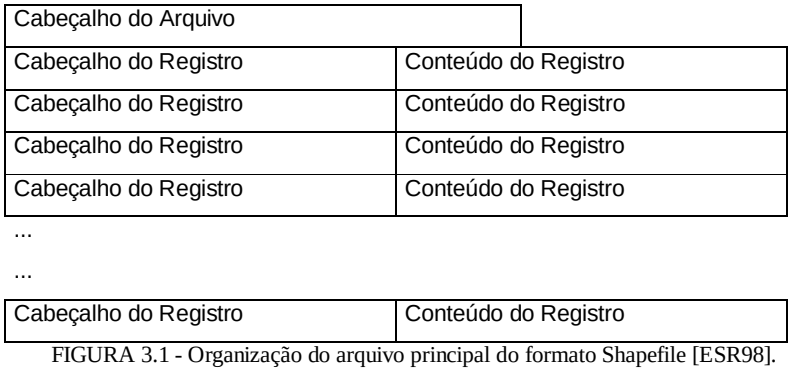

FIGURA 3.1 - Organização do arquivo principal do formato Shapefile [ESR98].

O cabeçalho do arquivo tem tamanho de cem bytes. A Tabela 3.1 mostra os campos que o compõem com sua posição, valor, tipo, e *byte order*. O campo *byte order* indica a ordem dos *bytes* em uma palavra de memória inteira. Pode ser de dois tipos: *little endian byte order* (o *byte* de menor ordem está armazenado no menor endereço de memória); *big endian byte order (*o *byte* de maior ordem está armazenado no menor endereço de memória).

TABELA 3.1 - Descrição do cabeçalho do arquivo principal do Shapefile [ESR98].

Posição	Campo	Valor	Tipo	Byte Order
Byte 0	File Code	9994	Integer	Big
Byte 4	Unused	0	Integer	Big
Byte 8	Unused	0	Integer	Big
Byte 12	Unused	0	Integer	Big
Byte 16	Unused	0	Integer	Big
Byte 20	Unused	0	Integer	Big
Byte 24	FileLength	File Length	Integer	Big
Byte 28	Version	1000	Integer	Little
Byte 32	Shape Type	Shape Type	Integer	Little
Byte 36	Bounding Box	Xmin	Double	Little
Byte 44	Bounding Box	Ymin	Double	Little
Byte 52	Bounding Box	Xmax	Double	Little
Byte 60	Bounding Box	Ymax	Double	Little
Byte 68*	Bounding Box	Zmin	Double	Little
Byte 76*	Bounding Box	Zmax	Double	Little
Byte 84*	Bounding Box	Mmin	Double	Little
Byte 92*	Bounding Box	Mmax	Double	Little

O Shapefile restringe que todas as feições pertencentes a um determinado arquivo devem ter o mesmo tipo de geometria ou geometria nula. Os tipos de geometria permitidos estão listados na Tabela 3.2.

O item *Bounding Box* no cabeçalho do arquivo principal determina o limite de valores dos eixos X e Y (e potencialmente M e Z) onde todas feições do arquivo estão contidas. Se o arquivo for vazio, os valores de Xmin, Ymin, Xmax, e Ymax não são

especificados. Mmin e Mmax podem conter o valor "*no data*" para tipos de geometria que não têm medidas.

O cabeçalho de cada registro armazena o número do registro e o tamanho de seu conteúdo, tem tamanho fixo de oito bytes. O conteúdo do registro armazena os dados geométricos para a feição que está sendo descrita. O tamanho do conteúdo depende do número de partes e vértices da feição. O tipo de geometria zero indica uma feição sem geometria.

TABELA 3.2 - Tipos de geometria do Shapefile [ERS98].

Valor	Tipo de Geometria
0	Formato Nulo
1	Point
3	Polyline
5	Polygon
8	MultiPoint
11	PointZ
13	PolylineZ
15	PolygonZ
18	MultiPointZ
21	PointM
23	PolylineM
25	PolygonM
28	MultiPointM
31	MultiPatch

3.1.1.1.1 Tipos de geometria no espaço X, Y

A descrição dos tipos de geometria suportados pelo formato Shapefile no espaço X,Y, e uma definição da classe que define esses tipos utilizando a sintaxe da linguagem C++, estão na Tabela 3.3.

TABELA 3.3 - Tipos de Geometria no espaço X,Y do Shapefile [ESR98]. (continua)

Tipo	Descrição	Definição

Point	Consiste de um par de coordenadas de tipo double na ordem X,Y.	Point
		{
		double X // Coordenada X
		double Y // Coordenada Y
		}

MultiPoint	Representa um conjunto de Point.	MultiPoint
		{
		double[4] Box
		// Envelope
		integer NumPoints
		// número de pontos
		Point[NumPoints] Point
		// Conjunto de pontos
		}

TABELA 3.3 - Tipos de Geometria no espaço X,Y do Shapefile [ESR98]. (continuação)

Tipo	Descrição	Definição

PolyLine	Uma Polyline é um conjunto ordenado de vértices que consiste de uma ou mais partes. Uma parte é uma seqüência de um ou mais pontos. Uma parte pode não estar conectada a outra e pode ou não haver uma intersecção.	PolyLine
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		}

Polygon	Consiste de um ou mais anéis. Um anel é um conjunto de quatro ou mais pontos conectados que formam um laço fechado e sem intersecção.Um Polygon pode ter vários anéis que são referenciados como suas partes.	Polygon
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		}

3.1.1.1.2 Tipos de geometria no espaço X, Y com medida

A descrição dos tipos de geometria suportados pelo formato Shapefile no espaço X,Y com medida, e uma definição da classe que define esses tipos utilizando a sintaxe da linguagem C++, estão na Tabela 3.4.

TABELA 3.4 - Tipos de Geometria no espaço X,Y com medida do Shapefile [ESR98]. (continua)

Tipo	Descrição	Definição

PointM	Consiste de um par de coordenadas de tipo double na ordem X,Y, mais uma medida M.	PointM
		{
		double X // Coordenada X
		double Y // Coordenada Y
		double M // Medida
		}

MultiPointM	Representa um conjunto de PointM.	MultiPointM
		{
		double[4] Box
		// Envelope
		integer NumPoint
		// número de pontos
		Point[NumPoints] Point

TABELA 3.4 - Tipos de Geometria no espaço X,Y com medida do Shapefile. (continuação)

Tipo	Descrição	Definição

		// Conjunto de pontos
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

PolyLineM	Uma PolylineM consiste de uma ou mais partes. Uma parte é uma seqüência de um ou mais pontos. Uma parte pode não estar conectada a outra e pode ou não haver uma intersecção.	PolyLineM
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

PolygonM	Consiste de um número de anéis. Um anel é um conjunto de quatro ou mais pontos conectados que formam um laço fechado e sem intersecção. Um Polygon pode ter vários anéis que são referenciados como suas partes. As intersecções são calculadas no espaço X,Y e não X,Y,M.	PolygonM
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

3.1.1.1.3 Tipos de geometria no espaço X, Y, Z

A descrição dos tipos de geometria suportados pelo formato Shapefile no espaço X,Y,Z, e uma definição da classe que define esses tipos utilizando a sintaxe da linguagem C++, estão na Tabela 3.5.

TABELA 3.5 - Tipos de Geometria no espaço X,Y,Z do Shapefile [ESR98]. (continua)

Tipo	Descrição	Definição

PointZ	Consiste de uma tripla de coordenadas de tipo double na ordem X,Y,Z, mais uma medida M.	PointZ
		{
		double X // Coordenada X
		double Y // Coordenada Y
		double Z // Coordenada Z
		double M // Medida
		}

MultiPointZ	Representa um conjunto de PointZ.	MultiPointZ
		{
		double[4] Box
		// Envelope
		integer NumPoint
		// número de pontos
		Point[NumPoints] Point
		// Conjunto de pontos
		double[2] ZRange
		// Limites das medidas de Z (mim e max)
		double[NumPoints] ZArray
		// Valores de Z
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

PolyLineZ	Uma PolylineZ consiste de uma ou mais partes. Uma parte é uma seqüência de um ou mais pontos. Uma parte pode não estar conectada a outra e pode ou não haver uma intersecção.	PolyLineZ
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		double[2] ZRange
		// Limites das medidas de Z (mim e max)
		double[NumPoints] ZArray
		// Valores de Z
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

TABELA 3.5 - Tipos de Geometria no espaço X,Y,Z do Shapefile. (continuação)

Tipo	Descrição	Definição

PolygonZ	Consiste de um número de anéis. Um anel é um conjunto de quatro ou mais pontos conectados que formam um laço fechado e sem intersecção. Um Polygon pode ter vários anéis que são referenciados como suas partes.	PolygonZ
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		double[2] ZRange
		// Limites das medidas de Z (mim e max)
		double[NumPoints] ZArray
		// Valores de Z
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

MultiPatch	Consiste de um número de caminhos de superfície. Cada caminho de superfície descreve uma superfície. Cada um destes caminhos é reconhecido como uma parte do MultiPatch, e o tipo de parte controla como a ordem dos vértices é interpretada. Uma parte pode ser dos seguintes tipos: Triangle Strip, Triangle Fan, Outer Ring, Inner Ring, First Ring e Ring.	MultiPatch
		{
		double[4] Box
		// Envelope
		integer NumParts
		// Número de partes
		integer NumPoints
		// Número total de pontos
		integer[NumParts] Parts
		// Índice do primeiro ponto de uma parte
		integer[NumParts] PartType
		// Tipo da parte
		Point[NumPoints] Points
		// Pontos de todas as partes
		double[2] ZRange
		// Limites das medidas de Z (mim e max)
		double[NumPoints] ZArray
		// Valores de Z para todos os pontos
		double[2] MRange
		// Limites das medidas (mim e max)
		double[NumPoints] MArray
		// Um array de medidas
		}

3.1.1.2 Arquivo de Índice

Um arquivo de índice contém um registro de cabeçalho com cem bytes de tamanho, seguido por registros de tamanho fixo de oito bytes. A Figura 3.2 ilustra a organização de um arquivo de índices.

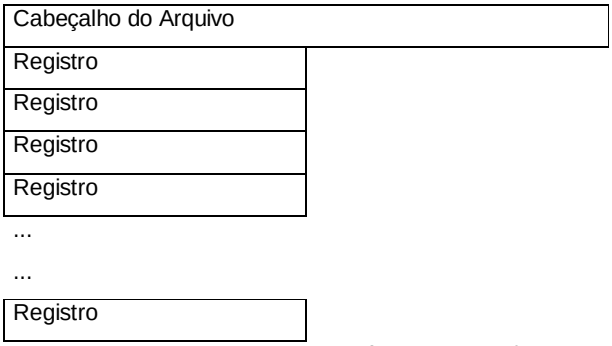

FIGURA 3.2 - Organização do arquivo de índice do Shapefile [ERS98].

O formato do registro cabeçalho do arquivo de índice é idêntico ao cabeçalho do arquivo principal descrito anteriormente. Os registros de índice armazenam o *offset* e o tamanho do conteúdo dos registros do arquivo principal. O *offset* de um registro no arquivo principal é o número de palavras de 16-bits do início do arquivo principal até o primeiro byte do cabeçalho do registro em questão. Por exemplo, o *offset* do primeiro registro do arquivo principal é cinqüenta, uma vez que o cabeçalho tem um tamanho fixo de cem bytes.

3.1.1.3 Arquivo de dados descritivos no formato dBASE

O arquivo de dados descritivos contém qualquer atributo das feições que seja necessário ou atributos chaves aos quais outras tabelas possam se relacionar. O formato é padrão de arquivos DBF usado por muitas aplicações baseadas em tabelas nos sistemas operacionais Windows™ e DOS. Mais informações sobre esse formato de arquivo podem ser encontradas em [BAC2000]. Qualquer conjunto de campos pode estar presente na tabela. Para cada feição do arquivo principal deve haver um registro na tabela de dados e os registros devem estar na mesma ordem deste arquivo.

3.1.2 GML (Geography Markup Language)

A GML é uma codificação em XML Schema para transporte e armazenamento de informações geográficas, incluindo aspectos geométricos e propriedades de feições geográficas. A GML foi desenvolvida pelo consórcio OpenGIS [OGC99] e baseia-se na especificação abstrata do mesmo, segundo a qual uma feição geográfica é definida como uma abstração de um fenômeno do mundo real; é uma feição geográfica se está associada com uma localização relativa a superfície terrestre. [OGC2001]

Assim, uma representação do mundo real pode ser entendida como um conjunto de feições. O estado de um feição é definido por um conjunto de propriedades, onde cada propriedade pode ser entendida como um conjunto de três elementos {nome, tipo, valor}. O tipo de feição determina as propriedades que a mesma pode possuir. Feições geográficas são aquelas que o valor de suas propriedades podem ser uma geometria. Uma coleção de feições é uma coleção que, pode por si mesmo, ser considerada como uma feição. Conseqüentemente, uma coleção de feições pode possuir propriedades próprias.

O modelo de feições utilizado pela GML, chamado “feições simples”, é uma simplificação do modelo descrito na especificação abstrata do OpenGIS [OGC99], mostrado na Figura 3.3.

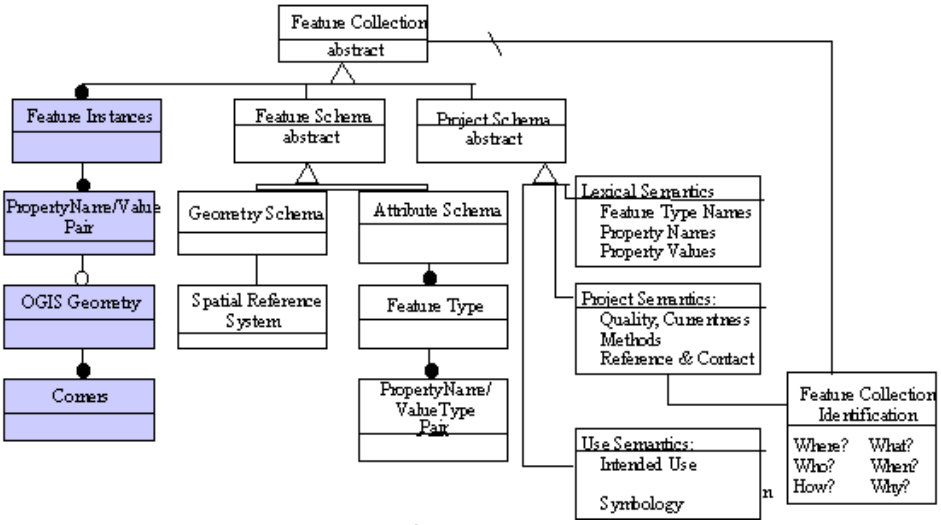

FIGURA 3.3 - Modelo de feições abstratas do OpenGIS [OGC2001].

Segundo esse modelo as feições são definidas para terem propriedades simples (boolean, integer, strings) ou propriedades geométricas. As geometrias estão definidas em um SRE (Sistema de Referência Espacial) de duas dimensões, sendo que curvas são obtidas por interpolação linear. A geometria de um objeto pode, ainda, ser representada por coleções de outros tipos de geometria: coleções homogêneas, como multipontos, multilinhas e multipolígonos, ou, ainda, por coleções heterogêneas. Nos dois casos, o SRE da geometria de mais alto nível é utilizado em todas as medidas.

3.1.2.1 Codificação em GML

Uma das alternativas para descrição de estrutura de documentos em XML são os esquemas. A GML, na versão 2.0, está de acordo com a recomendação XML *Schema* [XML00, XML00a] e consistente com a recomendação XML *Namespaces* [XML99]. *Namespaces* são usados para distinguir as definições de feições e propriedades definidas no domínio específico de uma aplicação para outra e das construções originais definidas nos módulos da GML. Um *namespace* é uma entidade conceitual identificada por um URL (*Uniform Resource Locator*), http://www.opengis.org/gml para o *namespace* central da GML, que denota uma coleção de nomes de elementos e definições de tipo.

A GML 2.0 define três esquemas básicos para a codificação de informações espaciais: esquema geométrico (geometry.xsd), esquema de feições (featrure.xsd), e esquema Xlink (xlinks.xsd). Este último fornece informações para que a aplicação suporte funcionalidades de ligação entre elementos. Os dois primeiros esquemas possibilitam a codificação de uma grande variedade de informações geoespaciais. Estes esquemas básicos podem ser vistos como pacotes distintos, com dependências, como ilustrado na Figura 3.4.

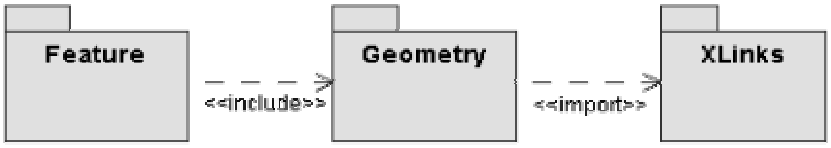

FIGURA 3.4 - Esquemas básicos da GML em pacotes[OGC2001].

Além dos esquemas básicos disponibilizados pela GML, é possível que sejam criados esquemas de aplicação, definidos pelo usuário para serem utilizados em um domínio particular. Várias regras devem ser observadas na criação de um esquema de aplicação para que este tenha validade.

Nas subseções a seguir são introduzidos os dois esquemas XML, geométrico e de feições.

3.1.2.1.1 O esquema de Feições

O esquema de Feições utiliza o elemento include para trazer as construções geométricas da GML e torná-las disponíveis para uso nas definições dos tipos de feições. A Figura 3.5 é uma representação UML do esquema de Feições. Neste esquema uma propriedade geométrica é modelada como uma classe de associação que liga uma feição com a sua geometria. O esquema de Feições também define tipos de elementos abstratos e concretos. O esquema de Feições da GML completo está definido em [OGC2001].

O elemento básico para descrever uma feição em GML é _Feature, a qual pode ou não ter propriedades geométricas. As feições geográficas que compõem a aplicação são definidas como subclasses dos tipos AbstractFeatureCollectionType ou AbstractFeatureType.

A GML 2.0 fornece, ainda, suporte a construção de coleções de feição. Para que um elemento de um esquema de aplicação seja considerado uma coleção de feições ele deve ser subclasse de AbstractFeatureCollectionType e declarado como substituto do elemento abstrato _FeatureCollection. Uma coleção de feições usa o elemento featureMember para definir as feições e/ou coleções de feição que ela contém.

Em uma coleção de feições um elemento featureMember pode conter uma feição simples ou apontar para uma feição que está armazenada remotamente, utilizando um conjunto específico de atributos definidos no esquema Xlink.

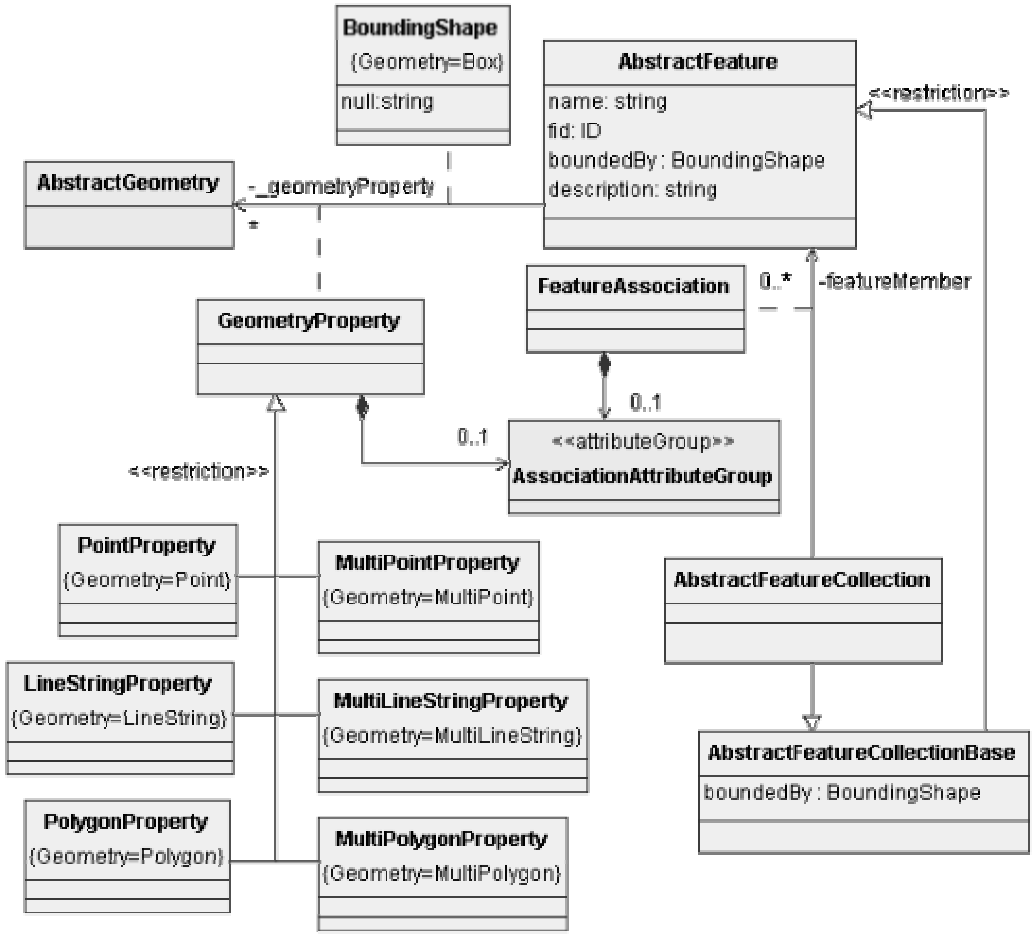

FIGURA 3.5 - Esquema de Feições em UML [OGC2001].

Tomemos como exemplo um tipo de feição sem geometria chamada Pessoa, que é definida por uma propriedade do tipo *string* chamada sobrenome e uma propriedade do tipo *integer* chamada idade. A feição Pessoa pode, ainda, ter zero ou mais ocorrências de uma propriedade de tipo *string* chamada apelido. Assim, uma instância da feição Maria pode ser definida em XML como segue:

```
<Pessoa>
        <nome>Maria</nome>
        <sobrenome>Silva</sobrenome>
        <idade>35</idade>
        <apelido>Dada</apelido>
        <apelido>Morena</apelido>
</Pessoa>
```

A definição de um esquema XML para suportar o exemplo acima seria:

```
<element name="Pessoa" type="ex:PessoaType"/>
<complexType name="PessoaType">
        <sequence>
                <element name="nome" type="string"/>
                <element name="sobrenome" type="string"/>
                <element name="idade" type="integer"/>
```

```
            <element name="apelido" type="string" minOccurs="0"
                        maxOccurs="unbounded"/>
        </sequence>
</complexType>
```

Entretanto, usando o esquema de Feições da GML, é necessário identificar quais elementos são feições e quais são propriedades. No exemplo citado acima, Pessoa é uma feição e idade é uma propriedade. Uma definição em GML para suportar o exemplo é a seguinte:

```
<element name="Pessoa" type="ex:PessoaType"
        substitutionGroup="gml:_Feature" />
<complexType name="PessoaType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
            <element name="nome" type="string"/>
            <element name="sobrenome" type="string"/>
            <element name="idade" type="integer"/>
            <element name="apelido" type="string" minOccurs="0"
                        maxOccurs="unbounded"/>
        </sequence>
      </extension>
    </complexContent>
</complexType>
```

A propriedade substitutionGroup definida no elemento Pessoa indica que este elemento pode substituir um elemento do tipo gml:_Feature. Ou seja, indica que Pessoa é uma feição. Enquanto, o rótulo <extension base> indica que o elemento PessoaType é um subtipo de gml:AbstractFeatureType. Portanto, é um tipo de feição.

3.1.2.1.2 O esquema Geométrico

O esquema Geométrico da GML tem definições de tipos para elementos abstratos; elementos geométricos concretos como ponto, linha e polígono; e tipos complexos compostos por um ou mais elementos geográficos. A Figura 3.6 é uma representação em UML do esquema Geométrico da GML. O esquema Geométrico da GML está definido em [OGC2001].

As classes AbstractGeometry e AbstractGeometryCollectionBase são classes abstratas que definem propriedades para todos os elementos da hierarquia. O atributo srsName indica o nome do sistema de referências utilizados, e o atributo gid armazena o identificador único da geometria.

As geometrias são descritas por pontos (Point), seqüências de linhas (LineString), um anel fechado de linhas (LinearRing), e polígonos (Polygon). Podem ser agrupadas em coleções (GeometryCollection), sendo especializadas em coleções de polígonos (MultiPolygon), de linhas (MultiLineString) e de pontos (MultiPoint). As geometrias podem, ainda, ser associadas livremente, de acordo com a semântica da aplicação (GeometryAssociation). Além destas classes também são definidas informações sobre coordenadas e retângulos, para melhor definir um objeto geográfico [FOR2002].

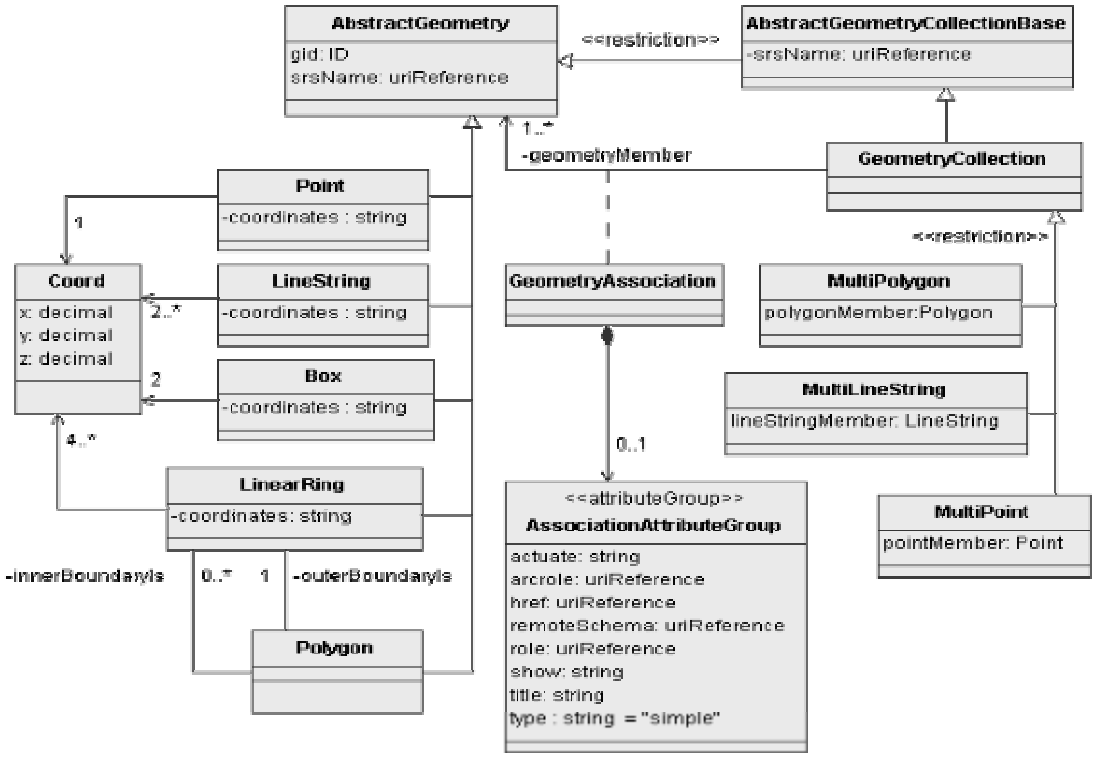

FIGURA 3.6 - Esquema Geométrico da GML [OGC2001].

As coordenadas para uma geometria são definidas em relação a algum SRE e podem ser representadas de duas formas. Através do elemento <coord>, que é definido como sendo composto pelos elementos X, Y e Z, sendo os dois últimos opcionais. E pelo elemento <coordinates>, composto por uma lista de coordenadas, compostas por dois ou três valores decimais.

O elemento **Point** é usado para codificar instâncias da classe geométrica Point. Cada elemento <Point> possui um elemento **<coord>** ou **<coordinates>** contendo exatamente uma tupla de coordenadas. O atributo srsName é opcional desde que o elemento Point esteja contido em outro elemento que especifique um sistema de referência. O atributo gid é opcional assim como em todos os outros tipos geométricos.

O elemento **Box** é utilizado para codificar extensões. Cada elemento **<Box>** possui uma conjunto de dois elementos **<coord>** ou **<coordinates>** contendo exatamente duas tuplas de coordenadas, a primeira tupla com os valores mínimos das medidas dos eixos e a segunda com os valores máximos. O atributo srsName é obrigatório, já que o elemento Box não pode estar contido em outros elementos.

Um elemento **LineString** é um caminho linear definido por uma lista de pontos que conectam-se diretamente, formando um segmento de linha. Um caminho fechado tem a primeira e a última coordenadas coincidentes. No mínimo duas coordenadas são requeridas.

Um elemento **LinearRing** é um caminho linear simples e fechado, o qual é definido por uma lista de pontos que conectam-se diretamente, formando um segmento de linha. A última coordenada deve ser idêntica a primeira e, no mínimo, quatro coordenadas são requeridas. Desde que este elemento seja usado na construção de um polígono, então o atributo srsName não é necessário.

Um elemento **Polygon** é um polígono fechado. Qualquer par de pontos em um polígono pode ser conectado a outro através de um caminho. Os limites de um elemento Polygon são definidos por elementos LinearRing. Existem uma distinção entre os limites exterior e interior, os elementos LinearRing que formam o limite interior não podem atravessar um ao outro e não podem conter nenhum outro. Deve existir um limite exterior e zero ou mais limites interiores.

Existem várias coleções de geometrias homogêneas que estão definidas no esquema geométrico. Um **MultiPoint** é uma coleção de Point; um **MultiLineString** é uma coleção de LineString; e um **MultiPolygon** é uma coleção de Polygon. Todas essas coleções usam propriedades apropriada para conter os elementos como seus membros. O atributo srsName deve aparecer somente na GeometryCollection mais externa. Além destas, existe também as coleções heterogêneas representadas pelo elemento **MultiGeometry**, este pode conter qualquer elemento de geometria primitiva tais como Point, Polygon, MultiPolygons e outros elementos MultiGeometry. O elemento MultiGeometry possui uma propriedade genérica chamada geometryMember que retorna o próximo elemento da coleção.

A GML 2.0 fornece um conjunto pré-definido de propriedades geométricas que podem ser usadas para relacionar geometria a tipos particulares de feição. A Tabela 3.6 mostra a relação desse conjunto de propriedades.

TABELA 3.6 – Propriedades Geométrica da GML 2.0 [FOR2002].

Nome Formal	Nome Descritivo	Tipo de Geometria
BoundedBy	-	Box
PointProperty	Location,position,centerOf	Point
LineStringProperty	CenterLineOf, edgeOf	LineString
PolygonProperty	ExtentOf, coverage	Polygon
GeometryProperty	-	qualquer geometria
MultiPointProperty	MultiLocation, multiPosition, multiCenterOf	MultiPoint
MultiLineStringProperty	MultiCenterLineOf, multiEdgeOf	MultiLineString
MultiPolygonProperty	Multiextentof, multiCoverage	MultiPolygon
MultiGeometryProperty	-	MultiGeometry

A seguir um fragmento de esquema que define a propriedade location para o exemplo Pessoa descrito anteriormente. Neste exemplo, através desta propriedade, foi definido uma geometria do tipo Point para o tipo de feição PessoaType.

```
<element name="Pessoa" type="ex:PessoaType"
      substitutionGroup="gml:_Feature" />
<complexType name="PessoaType">
   <complexContent>
    <extension base="gml:AbstractFeatureType">
     <sequence>
           <element name="nome" type="string"/>
           <element name="sobrenome" type="string"/>
           <element name="idade" type="integer"/>
           <element name="apelido" type="string" minOccurs="0"
                 maxOccurs="unbounded"/>
           <element ref="gml:location"/>
     </sequence>
```

```
    </extension>
  </complexContent>
</complexType>
```

É possível, também, definir uma propriedade geométrica específica de um esquema de aplicação. Para o exemplo acima, pode ser desejável especificar a localização de Pessoa com uma propriedade chamada localizaçãoPessoa e não location.

```
<element name="Pessoa" type="ex:PessoaType"
      substitutionGroup="gml:_Feature" />
<complexType name="PessoaType">
  <complexContent>
    <extension base="gml:AbstractFeatureType">
      <sequence>
            <element name="nome" type="string"/>
            <element name="sobrenome" type="string"/>
            <element name="idade" type="integer"/>
            <element name="apelido" type="string" minOccurs="0"
                  maxOccurs="unbounded"/>
            <element  name="localizaçãoPessoa"
                  type="gml:PointPropertyType" />
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

Neste exemplo, gml:PointPropertyType, que está definido no esquema geométrico da GML, está disponível como um tipo de propriedade pré-definida, da mesma forma que são os tipos *string* e *integer*.

3.1.2.1.3 Codificação de associações de feições

Existem duas maneiras de definir associações, por elementos contidos ou por ligações. Os elementos contidos são identificados nas coleções de feição, através da propriedade featureMember, descrito na Seção 3.2.1.1. Esse tipo de associação só pode ser utilizada em relacionamentos binários e não possui propriedades próprias.

Os atributos Xlink são utilizados, normalmente, para referência de feições que não estejam no mesmo documento XML, entretanto também é possível utilizar esse tipo de atributo para codificar relacionamentos que não possam ser expressos através de elementos contidos. Relacionamentos deste tipo podem, ou não, possuir propriedades próprias. Consideremos o relacionamento municipio_empresa, entre duas feições chamadas municipio e empresa. O esquema da aplicação para defini-lo é o seguinte:

```
<element name="Municipio" type="ex:MunicipioType"
      substitutionGroup="gml:_Feature" />
<element name="Empresa" type="ex:EmpresaType"
      substitutionGroup="gml:_Feature" />
<element name="municipio_empresa" type="ex:municipioEmpresaType"
      substitutionGroup="gml:_Feature" />
```

```
<complexType name="MunicipioType">
    <complexContent>
     <extension base="gml:AbstractFeatureType">
      <sequence>
            <element name="nome" type="string"/>
            <element name="populacao" type="integer"/>
            <element ref="gml:extentOf"/>
      </sequence>
     </extension>
    </complexContent>
</complexType>

<complexType name="EmpresaType">
    <complexContent>
     <extension base="gml:AbstractFeatureType">
      <sequence>
            <element name="RazaoSocial" type="string"/>
            <element name="cgc" type="integer"/>
            <element ref="gml:location"/>
      </sequence>
     </extension>
    </complexContent>
</complexType>

<complexType name="municipioEmpresaType">
    <complexContent>
     <extension base="gml:AbstractFeatureType">
      <sequence>
         <element name="producao" type="string"/>
         <element ref="ex:municipio" minOccurs="1" maxOccurs="1"  />
         <element ref="ex:empresa" minOccurs="1" maxOccurs="1" />
      </sequence>
     </extension>
    </complexContent>
</complexType>
```

O esquema do exemplo acima pode ser codificado em GML da seguinte forma:

```
<Municipio fig="mun001">
      <nome>Caxias do Sul</nome>
      <populacao>400000</populacao>
      <gml:extentOf>...</ gml:extentOf>
</Municipio>

<Empresa fig="emp001">
      <razaosocial>Randon Implementos S.A</razaosocial>
      <cgc>88610829000157</cgc>
```

```
        <gml:location>...</ gml:location>
</Empresa>

<municipio_empresa fig="rel001">
        <producao>lonas de freio</producao>
        <municipio xlink:type="simple" xlink:href="#mun001"/>
        <empresa xlink:type="simple" xlink:href="#emp001"/>
</municipio_empresa>
```

3.1.3 SAIF (Spatial Archive and Interchange Format)

O SAIF é um padrão que define um modelo para especificação e troca de dados espaciais independente de fornecedor, baseado no paradigma de orientação a objetos. Este padrão é proposto pelo *Survey and Resource Mapping Branch - Ministry of Environment Lands and Parks* do Canadá. O SAIF define uma notação, chamada OSN (*Object Syntax Notation*), utilizada para a definição das instâncias específicas das construções do mundo real, isto é, a descrição atual dos dados dos objetos de interesse. Uma segunda notação, chamada CSN (*Class Syntax Notation*), permite a definição das construções espaciais e temporais dos dados. O conjunto destas duas notações é chamado de SAIFtalk.

O SAIF é dividido em três níveis de abstração, ilustrados na Figura 3.7. O primeiro nível, chamado Modelo de Dados, é o mais abstrato. Neste nível toda a informação é definida como construções matemáticas, incluindo objetos abstratos, listas, enumerações, e primitivas. No segundo nível, chamado Esquema Padrão SAIF, são definidas as construções espaciais e temporais que compõem o núcleo do esquema padrão do SAIF. São mais de 300 construções básicas. A partir das construções do esquema padrão do SAIF é possível representar as entidades do mundo real através de tipos e classes adicionais que constituem o esquema do usuário, terceiro nível de abstração.

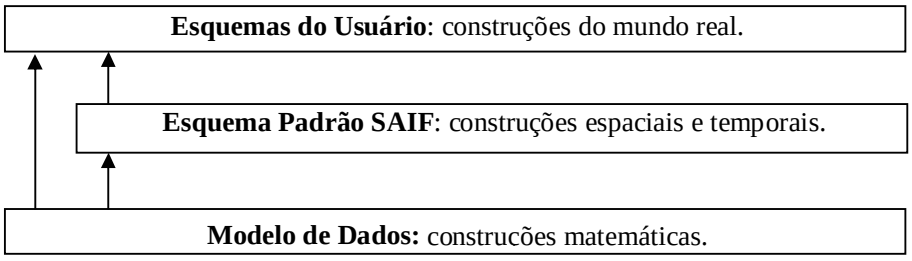

FIGURA 3.7 – Paradigma de modelagem em multi-camadas do SAIF [BRI95].

3.1.3.1 Modelo de Dados

O modelo de dados do SAIF está organizado em uma hierarquia de tipos, conforme ilustrado na Figura 3.8. Este modelo serve como base para o desenvolvimento de novas classes e enumerações. Todas as classes fornecidas pelo esquema padrão do SAIF ou dos esquemas definidos pelo usuário são subclasses de AbstractObject diretamente ou através de uma de suas subclasses. Assim como enumerações são derivadas de Enumeration. As classes Collection e Primitive não podem ser especializadas, uma vez que são usadas apenas como domínio de atributos.

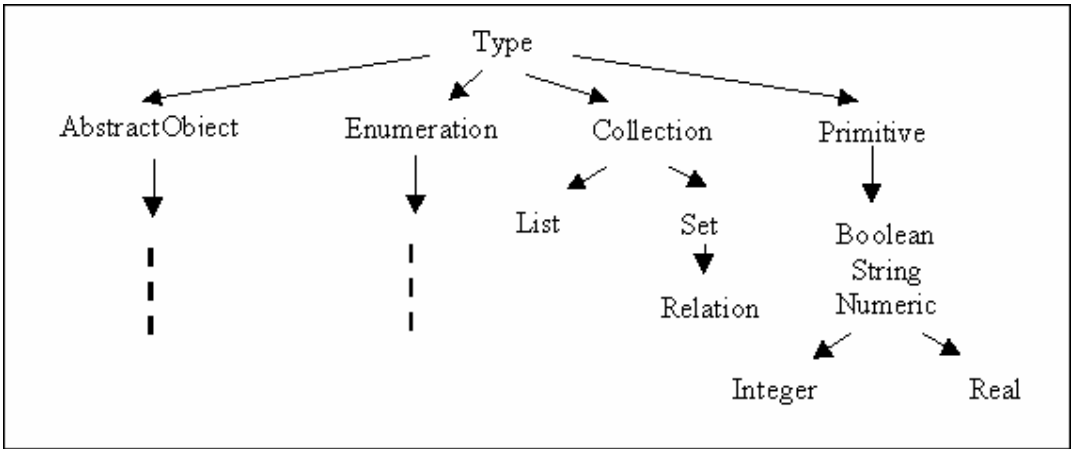

FIGURA 3.8 – Modelo de Dados do SAIF [BRI95].

3.1.3.2 Esquema Padrão SAIF

Em um alto nível de abstração, o SAIF distingue entre representações dos fenômenos do mundo real e a representação do espaço e tempo nos quais eles existem. Representações de fenômenos do mundo real são definidos como objetos geográficos e são representados por especializações da classe GeographicObject. Tais objetos podem ter relacionamentos entre si, incluindo relacionamentos espaciais e temporais.

A posição de cada fenômeno no espaço é representada por seu objeto espacial associado. Se espaço e tempo são considerados, a posição de fenômenos é representada por objetos espaço-temporais. Assim, se um tipo particular de objeto geográfico é considerado como espacial na natureza, ele terá um objeto espacial correspondente. Se ele for considerado como espaço-temporal na natureza, terá um objeto espaço-temporal correspondente.

O esquema padrão fornece um grande número de construções espaciais e espaço-temporais, muitas das quais podem ser subclasses para os usuários definirem novas classes, que serão diretamente usadas por eles. Apenas um subconjunto destas construções é relevante para os objetivos deste trabalho. A Figura 3.9 representa, graficamente, na notação UML, as classes e associações que compõem este subconjunto do esquema padrão do SAIF.

Parte da hierarquia de GeographicObject e SpatialObject são descritas nas subseções abaixo, uma vez que estas classes possuem correspondência aos conceitos das classes do *framework* GeoFrame e serão utilizadas posteriormente no mapeamento do padrão SAIF para o GeoFrame.

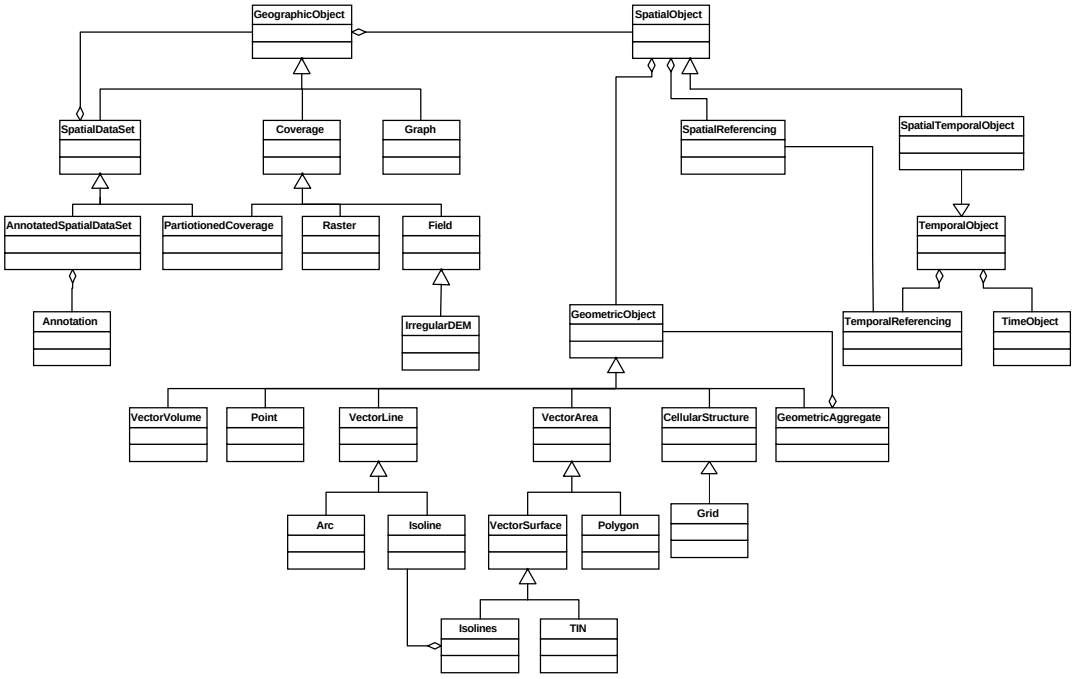

FIGURA 3.9 – Esquema Padrão do SAIF (parcial).

3.1.3.2.1 GeographicObject

Um objeto pertencente a classe GeographicObject representa alguma parte do mundo real. Este objeto possui uma posição no espaço e possivelmente no tempo, e é representado como um membro das classe SpatialObject ou SpatioTemporalObject. Existem quatro casos em que os objetos geográficos podem ser reconhecidos.

No primeiro caso, o objeto geográfico corresponde a um fenômeno no mundo real. Normalmente o usuário define subclasses de GeographicObject segundo suas necessidades. Por exemplo, o usuário pode definir uma subclasse de GeographicObject chamada Rodovias. Objetos do tipo Rodovias provavelmente tem sua geometria descrita por linhas ou polígonos. Este tipo de objeto geográfico corresponde a um fenômeno concreto do mundo real. Entretanto, um objeto geográfico pode corresponder a um fenômeno artificial, como uma jurisdição policial ou um distrito escolar.

O segundo caso pertence ao conjunto de dados espaciais, coleções de dados referenciados espacialmente, que são agrupados por razões de conveniência. Este caso refere-se a especialização da classe GeographicObject chamada SpatialDataSet. Esta classe possui uma associação de agregação com a classe GeographicObject indicando que membros desta classe contém uma série de objetos geográficos. Tais objetos são considerados pertencentes ao mesmo conjunto de dados espaciais, geralmente devido a similaridade em metadados ou a referência espacial. Por exemplo, uma bacia hidrográfica pode ser definida com uma subclasse de SpatialDataSet que contenha sub-bacias. Cada sub-bacia pode ser, também, uma subclasse de SpatialDataSet composta por zonas de ecossistema, que é uma subclasse de GeographicObject.

O terceiro caso refere-se a classe Coverage. Objetos desta classe descrevem algum fenômeno distribuído sobre uma região arbitrária do espaço, podendo modelar fenômenos contínuos, como altitude, ou discretos, como uso do solo. Têm a função de mapear uma posição no espaço em um valor, através de uma função espacial. Uma função espacial tem sempre como domínio um conjunto de objetos da classe SpatialObject porém seu contra-domínio pode ser qualquer conjunto, numérico ou de objetos. Como o SAIF não oferece suporte para a definição de métodos ou funções, apenas os aspectos estruturais dos dados podem ser modelados. A classe Coverage possui três subclasses, que podem ser especializadas pelo usuário: raster, field e partitionedCoverage.

A classe raster é utilizada para descrever dados *raster* típicos, onde os valores são associados a cada posição em um *grid*. A grade é definida em termos de estruturas regulares, multi-dimensional, retangular, estrutura hexagonal ou triangular.

A classe field descreve a distribuição de um fenômeno no espaço, cuja geometria define uma superfície contínua, tal como: isolinhas ou modelo de elevação digital, rede triangular irregular (TIN), grade de pontos e pontos irregulares. Esta classe é especializada em IrregularDEM, que é capaz de descrever uma modelagem através de um conjunto de pontos distribuídos irregularmente sobre uma região geográfica.

A classe PartitionedCoverage consiste de uma série de objetos geográficos discretos, os quais podem ser desta mesma classe ou de outras. É utilizada, por exemplo, para definir áreas de censu, ou combinações de objetos discretos como rodovias e estradas de ferro. Objetos dessa classe não necessariamente preenchem todo o espaço geográfico.

O quarto caso preocupa-se com gráficos, incluindo redes. Gráficos consistem de um série de objetos geográficos que são conectados a outros objetos geográficos. Este caso refere-se a classe Graph, onde cada objeto desta classe consiste em um conjunto de conexões entre objetos da classe GeographicObjects, definidas como relacionamentos. Todos os objetos devem ser contínuos no espaço ou no tempo. Conexões de linhas aéreas são um exemplo de uso desta classe.

3.1.3.2.2 SpatialObject

Um objeto espacial representa uma região do espaço. O espaço é definido através da agregação com a classe SpatialReferencing, ou seja, é definida a localização do objeto em um SRE. Enquanto que a região é definida através de agregações com a classe geometricObject, que representa a geometria do objeto, sua forma.

3.1.3.2.3 GeometricObject

O SAIF considera objetos geométricos abstratos (pontos, arcos, polígonos), como sendo membros de uma classe de objetos conhecida como GeometricObject. Uma coleção arbitrária destes objetos geométricos pode ser considerada um objeto geométrico por si só. Isto inclui estruturas de vetor com dimensões zero, um, dois e três, assim como estruturas de grades retangulares, hexagonal, ou triangular com duas, três ou quatro dimensões. Os membros da classe GeometricObject são especificados através de um ou mais pontos, cada um dos quais dando uma das quatro coordenadas das dimensões (c1,c2,c3,t).

A classe GeometricObject está especializada em Point, VectorVolume, CellularStructure, VectorLine, VectorArea, GeometricAggregate.

Point é usada para definir a posição de um ponto na superfície terrestre.

CellularStructure representa uma estrutura de grade espacial ou espaço-temporal. Essa classe é especializada em Grid, que suporta estruturas de grade do tipo retangular, triangular e hexagonal.

VectorLine representa objetos geométricos com uma dimensão, sendo definidos espacialmente através de coordenadas. É especializado em Arc, que define um conjunto de dois ou mais pontos conectados a uma lista, e Isoline.

VectorArea representa objetos geométricos de duas ou três dimensões (VectorSurface) no formato de vetores. Esta classe é especializada em Polygon e VectorSurface.

GeometricAggregate representa a agregação de várias representações de objetos geométricos.

3.1.3.2.4 SpatialReferencing

Para que os valores das coordenadas c1, c2 e c3, de um determinado objeto geométrico, tenham significado no mundo real, a referência espacial deve ser fornecida. A primeira função de um objeto de referência espacial é fornecer informações sobre o sistema de coordenadas. O sistema de coordenadas refere-se ao modo com que coordenadas posicionais são localizadas com respeito a alguma superfície planar ou elipsóide.

Em muitos casos, o sistema de coordenadas precisa somente especificar que as coordenadas estão localizadas em uma superfície retangular plana e não tem relação com o posicionamento na terra. Formas mais complexas de referência espacial definem o relacionamento entre localização de coordenadas e a posição da terra fornecendo o *datum* vertical e horizontal. O *datum* é um ponto onde a superfície do elipsóide de referência, usado na projeção da superfície terrestre, toca a terra.

3.1.3.2.5 SpatioTemporalObject

Um objeto espacial é composto de um objeto geométrico e uma referência espacial. Um objeto temporal é composto de um objeto tempo e uma referência temporal. A classe SpatioTemporalObject é definida como uma especialização de SpatialObject e TemporalObject. Conseqüentemente, um membro da classe SpatioTemporalObject inclui objetos pertencentes as seguintes classes: GeometricObject, SpatialReferencing, TemporalReferencing, e TimeObject.

Os usuários não podem definir novas subclasses de SpatioTemporalObject, TemporalObject ou TimeObject.

3.1.3.2.6 TimeObject

Um membro desta classe possui um valor para horário, data, um intervalo, uma duração, ou uma coleção destes valores. Datas e harários de 24 horas podem ser usados separadamente ou juntos. Dois tipos de intervalo são reconhecidos, intervalos de mês e ano e intervalos de horas e dias. O SAIF adicionou a noção de duração, que é definido

por um ponto inicial no tempo e um intervalo estendido além deste ponto. O SAIF também separa o valor de um objeto tempo de sua referência temporal.

3.1.3.2.7 TemporalReferencing

Referência temporal deve ser fornecida quando são dados valores em TimeObject ou as classes de coordenadas incorporadas ao tempo. Referência temporal fornece informação quando:

- ✓ A Coordenada Universal de Tempo (CUT) está sendo usada e se o tempo de GPS é aplicado (nos caso em que os saltos de segundos são ignorados);
- ✓ A compensação do CUT é em horas e minutos.

3.1.3.2.8 Annotation

Informações espaciais freqüentemente incluem textos e símbolos. Textos podem ser usados como um rótulo. Pode também se referenciar a uma determinada região com um limite definido, como “Oceano Pacífico”. Símbolos podem ser usados para indicar aeroportos, cidades históricas, etc. A definição de textos ou símbolos incluem, no mínimo, um ponto para indicar onde o texto deve ser colocado. Nos casos em que o texto fica sobre uma curva ou em múltiplas linhas, uma série de pontos deve ser fornecido.

Textos e símbolos devem também ser espacialmente referenciados. Um objeto desta classe inclui um objeto da classe TextOrSymbolObject, assim como um objeto da classe SpatialReferencing.

3.1.3.3 Esquema do Usuário

O usuário pode definir um esquema com novos tipos de dados para descrever a semântica das entidades e dos fenômenos do mundo real. Os fenômenos definidos pelo usuário são modelados como subclasses do Esquema Padrão do SAIF, desde que as classes possam ser especializadas.

3.1.3.4 CSN (Class Syntax Notation)

A CSN é uma notação utilizada para descrever as definições das classes do SAIF, é baseada na gramática BNF (*Forma Backus-Naur*), que foi estendida para atender às necessidades dos objetos espaciais. A notação completa da CSN/BNF e uma definição completa das regras de definição da CSN podem ser encontradas em [BRI95]. A Figura 3.10 ilustra a notação de definição de uma classe utilizando CSN.

```
<Superclass::NameSchema1
  subclass:        subclass::NameSchema2
  subclassing:     “o usuário pode definir suas subclasses aqui”
  attribute:       atributoX   DomínioAtributoX
  defaults:        atributoX   ValorDefaultAtributoX
  restricted:      AtributoX   RestriçãoAtributoX
  Constraints:     “regra de restrição”
  Comments:        “comentarios”
```

FIGURA 3.10 – Definição de classe em CSN.

A definição da classe fica inclusa em uma par < > para indicar o início e fim da definição. Essa notação permite a utilização de aspas para documentar comentários a respeito da classe.

Superclass - corresponde à definição da superclasse. Se esta não faz parte do Modelo de Dados ou do Esquema Padrão do SAIF, o nome da superclasse deve ser seguido por dois pontos e pelo nome do esquema do usuário onde está definida.

Subclass - especifica o nome da subclasse que é especialização de superclass, juntamente com o nome de um esquema externo, se for necessário. O nome das classes e o nome do esquema iniciam com letra maiúscula e podem ser seguidos por letra maiúsculas ou minúsculas, digitos numéricos e sublinhas (_).

Subclassing – utilizada para declara se a superclasse do Esquema Padrão do SAIF pode ser especializada pelo usuário, uma vez que, nem todas as superclasses do padrão podem ser estendidas.

Attributes – são as propriedades da classe. Cada um apresenta um tipo de domínio específico, podendo ser *primitive*, *enumeration*, *collection* ou *class*. O valor de um atributo pode ser opcional ou obrigatório, o que é indicado através da presença de [ ] ou { } respectivamente.

Defaults – é definido como o conteúdo inicial de um atributo. Caso não seja atribuído um valor ao atributo, o valor definido em *default* será utilizado se o atributo for obrigatório.

Restricted – corresponde às definições que restringem ou limitam o domínio dos atributos. Um atributo pode ser limitado tanto por um tipo de domínio quanto por um intervalo de valores. Esta cláusula pode ser utilizada para limitar a geometria do objeto.

Comments – permite especificar comentários para as classes.

A Figura 3.11 apresenta um exemplo de definição, de algumas classes do Esquema Padrão SAIF, utilizando a notação CSN.

```
<AbstractObject
subclass: GeographicObject
attributes: position SpatialObject
>
<GeographicObject
subclass: SpatialDataSet
attributes: [geoComponents] Set(GeographicObject)
>
<SpatialDataSet
subclass: AnnotatedSpatialDataSet
attributes: [annotationComponents] Set(Annotation)
>
<GeographicObject
subclass: Coverage
>
<SpatialDataSet,Coverage
subclass: PartitionedCoverage
>
<AbstractObject
subclass: SpatialObject
attributes: [geometry] GeometricObject
[spatialReferencing] SpatialReferencing
>
```

FIGURA 3.11 – Exemplo de definição de classes em CSN.

No exemplo, acima, a classe GeographicObject é definida como uma especialização da classe AbstractObject e possui um atributo chamado position do tipo SpatialObject. Como SpatialObject também é uma classe, isso indica uma associação entre as classes.

SpatialDataSet é uma especialização de GeographicObject. O atributo geoComponents está entre colchetes, indicando que é opcional. Seu tipo é a classe GeographicObject, como existe a expressão SET em sua definição, indica que possui uma associação de agregação com esta classe.

A definição da classe partitionedCoverage é um exemplo de herança múltipla. Esta classe é especialização de SpatialDataSet e Coverage.

3.2 Introdução ao *Framework* GeoFrame

As aplicações geográficas possuem características específicas que dificilmente são contempladas pelos modelos de dados conceituais de propósito geral. É preciso dar suporte às características espaciais das entidades geográficas, a seus aspectos temporais, à qualidade dos dados, às múltiplas representações espaciais e a limites não bem definidos, entre outros. Existem vários modelos conceituais para banco de dados geográficos propostos pela literatura. A maioria deles é baseada na orientação a objetos ou no modelo entidade-relacionamento. Neste contexto, pode-se citar: GeoOOA, Modul-R, GMOD, Geo-ER, e GeoIfo [LIS97].

Em [LIS99] é proposto o GeoFrame, um *framework* conceitual orientado a objetos, que fornece um diagrama de classes básicas para auxiliar o projetista tanto na modelagem conceitual dos dados geográficos como, também, na especificação de padrões de análise em bancos de dados geográficos. O GeoFrame não é um modelo conceitual de dados, é um diagrama de classes desenvolvido para ser usado como ponto de partida na modelagem conceitual de aplicações geográficas.

Em [ROC2001] são apresentadas alterações ao *framework* conceitual GeoFrame. Trata-se da versão 2.0 baseada nas regras do formalismo de orientação a objetos para utilização na modelagem conceitual de dados de aplicações geográficas. O GeoFrame utiliza a notação do diagrama de classes da Linguagem de Modelagem Unificada – UML [BOO2000]. O objetivo de um *framework* conceitual é o de fornecer um diagrama de classes que pode ser usado como base para a modelagem das classes de um domínio da aplicação, no caso, de aplicações geográficas. Além disso, o GeoFrame pode ser utilizado na especificação de padrões de análise em banco de dados geográficos. Nas subseções seguintes revisa-se o *framework* conceitual GeoFrame.

O GeoFrame foi escolhido para servir de base aos modelos que serão criados e mantidos pelo processo de engenharia reversa proposto, neste trabalho, por se tratar de uma representação genérica, independente de aplicação, extensível, e capaz de representar todos requisitos necessários para os modelos de dados inferidos pelo sistema de engenharia reversa aqui proposto.

3.2.1 Pacote Tema

A grande quantidade de tipos de dados diferentes que são manipulados em SIG, leva a utilização de algum tipo de classificação que reuna os objetos com alguma afinidade funcional ou estrutural em um determinando grupo. Esta maneira de

classificar os objetos da realidade, simplifica a forma de pensar e tratar os dados, e faz parte do cotidiano da comunidade de SIG, é o Tema ou *layer* como é implementado em alguns *software*. Conforme Lisboa [LIS2000], o uso de temas permite ao projetista, dividir o esquema de dados em subesquemas coesos, nos quais são agrupadas classes que estão fortemente relacionadas entre si. Por exemplo, o tema Hidrografia, reúne os objetos relacionados com os fenômenos hídricos da natureza como: rios, lagos, nascentes, corredeiras, etc. Já o tema TransporteTerrestre envolve as rodovias, ferrovias, pontes, estações e outros objetos que se relacionam ao assunto transporte por via terrestre [ROC2001].

No GeoFrame, o Tema é tratado como um pacote que reúne conceitualmente, por algum tipo de semelhança, objetos da realidade que se relacionam entre si. A figura 3.12 mostra os componentes de nível lógico que formam o *framework* conceitual GeoFrame, onde o pacote TEMA pode ser formado recursivamente por outro pacote TEMA e/ou pelo pacote PGEOFRAME. O pacote PGEOFRAME é constituído pela hierarquia de classes que serve para ser usada como base para a modelagem das classes do domínio da aplicação geográfica.

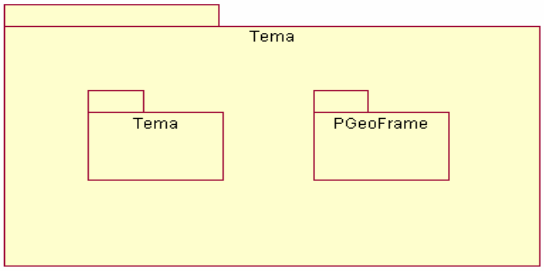

FIGURA 3.12 – Arquitetura lógica do *framework* conceitual GeoFrame [ROC2001].

3.2.2 Pacote PGeoFrame

A Figura 3.13 apresenta o pacote PGeoFrame com sua hierarquia de classes.

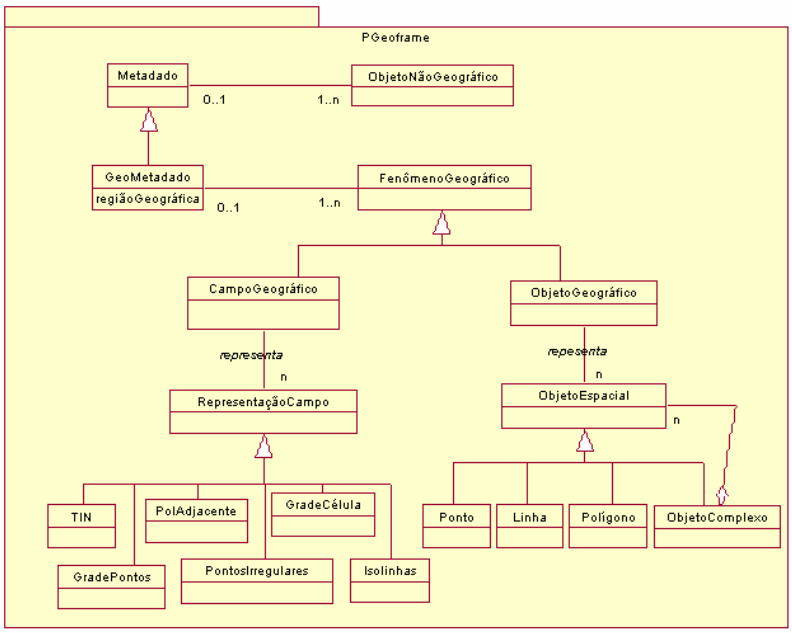

FIGURA 3.13 – Pacote PGeoFrame, com a hierarquia de classes que formam o GeoFrame [ROC2001].

3.2.3 Metadado e GeoMetadado

A necessidade de manter metadados sobre os dados da aplicação é inquestionável para sistemas georeferenciados, já que muitas das operações e análises que podem ser feitas com o dado espacial dependem de informações como a escala de aquisição deste dado, ou se o mesmo provêm de uma operação de generalização ou não. Portanto, todo objeto georeferenciado deve ter metadados que indiquem a sua origem e qualidade.

A Classe Metadado está associada a classe ObjNãoGeográfico. A Classe GeoMetadado é uma especialização da classe Metadado e está associada a FenômenoGeográfico.

Os atributos destas classes dependem do tipo de metadado que é necessário à aplicação que está sendo modelada.

3.2.4 FenômenoGeográfico e ObjNãoGeográfico

Todos os fenômenos do banco de dados geográfico que possuírem referência a sua localização geográfica na superfície da terra compõe a classe FenômenoGeográfico. Se um determinado fenômeno tem referência a superfície terrestre, mas isto não interessa para o banco de dados em questão, este deve ser considerado como parte da classe ObjNãoGeográfico.

A classe ObjetoNãoGeográfico compreende aqueles dados que não possuem uma referência espacial. Nem todos os dados que compõe o banco de dados geográficos são referentes a fenômenos geográficos. Existem vários dados convencionais que podem ser armazenados.

3.2.5 CampoGeográfico e ObjetoGeográfico

A classe FenômenoGeográfico é especializada nas classes CampoGeográfico e ObjetoGeográfico. Permitindo assim que o projetista consiga separar em classes distintas os objetos na visão de campo e objeto.

A classe ObjetoGeográfico é composta por todas as classes dos objetos geográficos percebidos na visão de objetos. Ou seja, aquelas entidades individuais, bem definidas e identificáveis. Cada entidade ocupa um determinado lugar no espaço. São exemplos deste tipo de classe: logradouros, distritos, e rios.

A classe CampoGeográfico é composta por todas as classes dos objetos geográficos que sejam percebidos pela visão de campo. Objetos que possuam uma distribuição contínua no espaço. Essas podem ter a distribuição discreta, como é o caso da população, ou distribuição contínua, como é o caso do tipo de solo.

3.2.6 ObjetoEspacial

A classe ObjetoEspacial generaliza as classes necessárias para a especificação da representação do componente espacial dos fenômenos geográficos percebidos na visão de objetos. São elas: Ponto, Linha, Polígono e ObjetoComplexo.

O GeoFrame utiliza pontos, linhas e polígonos para determinar a forma abstrata com que os fenômenos espaciais serão representados. A estrutura de dados que será utilizada para armazenar esses objetos espaciais não importam nessa fase do projeto.

Aqui, por exemplo, só importa determinar que um rio terá representação através de linhas sem se importar se isso será armazenado através de arcos ou de uma estrutura vetorial.

A classe ObjetoComplexo representa fenômenos espaciais que podem ser compostos por mais de um objeto espacial, por exemplo um arquipélago.

Dependendo da escala de visualização e do propósito o mesmo objeto geográfico pode ser mostrado em mais de um modelo descrito acima. Em uma determinada situação, por exemplo, um rio pode ser representado através de um segmento de linha e em outra situação o mesmo rio pode ser representado por um polígono.

3.2.7 RepresentaçãoCampo

A classe RepresentaçãoCampo é especializada em seis subclasses, são elas: GradeCélula, PolAdjacente, Isolinhas, GradePontos, TIN (rede triangular irregular) e PontosIrregulares. Segundo [LIS99] essas especializações representam os modelos espaciais que são adequados a modelagem de fenômenos na visão de campo.

Os seis modelos serão implementados no SIG através do modelo de representação matricial ou através do modelo vetorial. A escolha do modelo dependerá da escolha do projetista. Embora ambos possam ser usados para qualquer um dos casos.

3.2.8 Estereótipos

Com a intenção de melhorar a legibilidade do modelo são utilizados estereótipos, que é um mecanismo de simplificação do esquema. Na Figura 3.14 são mostrados os estereótipos utilizados para substituir os relacionamentos de generalização entre as classes do domínio e as classes do GeoFrame.

Os estereótipos indicam que o objeto é um ObjetoGeográfico ou um CampoGeográfico e a forma como ele é representado. A ausência de estereótipos indicam que o objeto é não espacial.

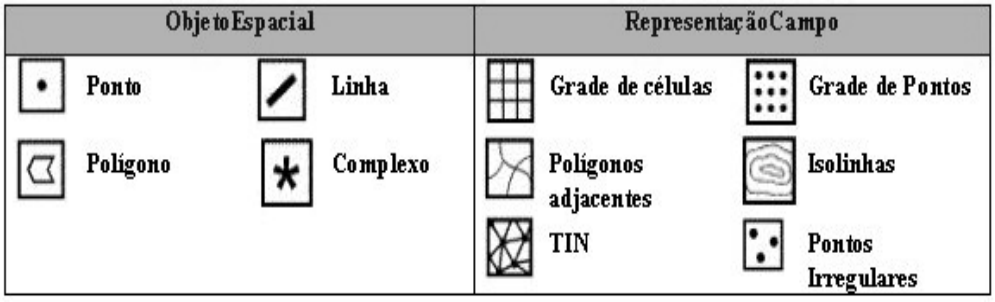

FIGURA 3.14 - Estereótipos espaciais do GeoFrame [ROC2001].

3.2.9 Exemplo de modelagem utilizando GeoFrame

A Figura 3.15 apresenta um exemplo de modelagem utilizando o *framework* GeoFrame com estereótipos. As classes Município e Distrito compõem a pacote PGeoFrame do tema Unidades Administrativas. Ambas são subclasses da classe ObjetoGreográfico representadas por Polígono.

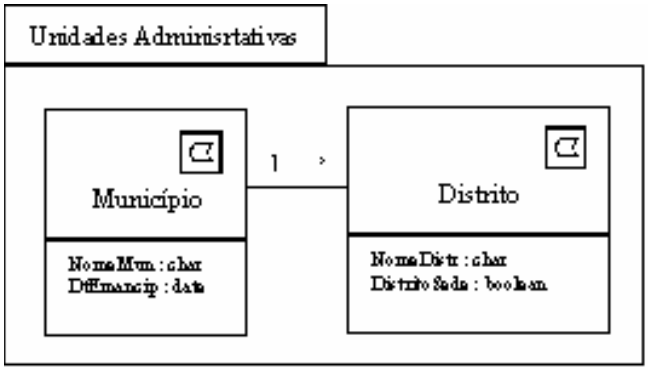

FIGURA 3.15 – Exemplo de modelagem com *framweork* GeoFrame [MAT2000].

3.3 Regras de mapeamento

Foram definidas regras que possibilitam o mapeamento dos objetos armazenados em um dos formatos de transferência de dados geográfico estudas para o *framework* GeoFrame. Cada objeto armazenado representa uma classe da aplicação que, por sua vez, representam especializações das classes do GeoFrame.

3.3.1 Regras de Mapeamento do Shapefile para GeoFrame

O principal ponto a ser considerado ao analisar um arquivo no formato Shapefile é a restrição de que, em um mesmo arquivo, apenas possam existir feições com o mesmo tipo de geometria (point, polyline, etc.). Normalmente, um banco de dados geográfico é organizado em camadas, cada uma com um tipo de tema (rios, edificações, rodovias, etc.). Sendo assim, ao se adquirir os dados de uma determina região geográfica o usuário irá receber um conjunto de arquivos, cada qual com um tema ou subtema. Por exemplo, uma carta topográfica seria estruturada da seguinte forma:

- ✓ Camada Rios – linhas;
- ✓ Camada Edificações – polígonos (ou pontos, depende da escala);
- ✓ Camada Limite Urbano – polígonos;
- ✓ Camada Rodovias – linhas;
- ✓ Camada Escolas – pontos.

Este formato de transferência de dados não possibilita a extração de muitas informações para seu mapeamento em um esquema GeoFrame. Para cada arquivo analisado é possível identificar se o tipo de dado é um objeto não geográfico ou um fenômeno geográfico, sua geometria, e atributos descritivos. Ou seja, apenas uma classe é mapeada para o GeoFrame. Não é possível identificar os relacionamentos entre os tipos de dados, pois estão em arquivos separados. O nome do tipo de dado deve ser informado pelo usuário ou extraído do nome do arquivo. O arquivo de índice não é

relevante para o mapeamento em questão. O usuário também deve escolher a que pacote Tema do esquema o tipo de dado deve ser agregado.

Identifica-se através do registro de cabeçalho do arquivo principal o tipo de geometria que está sendo importado. Se o tipo for igual a zero significa que estamos trabalhando com uma feição sem geometria, o que permite defini-la como subclasse da classe ObjetoNãoGeográfico do GeoFrame. Se o tipo for diferente de zero significa que a feição tem geometria e pode ser considerada subclasse da classe FenômenoGeográfico. O tipo de geometria irá determinar se a feição é subclasse de CampoGeográfico ou ObjetoGeográfico e a que especialização de RepresentaçãoCampo ou ObjetoEspacial, respectivamente, estará associado. A Tabela 3.7 mostra a relação entre os tipos de geometria, encontrados no formato Shapefile, e suas representação no GeoFrame.

TABELA 3.7 - Tipos de geometria e sua representação no GeoFrame.

Tipo de Geometria no Shapefile (valores de acordo com tabela 3.2)	**Classe no GeoFrame que será subclasse**	**Especialização de RepresentaçãoCampo ou ObjetoEspacial**
1, 11, 21	ObjetoGeográfico	Ponto
3, 13, 23	ObjetoGeográfico	Linha
5, 15, 25	ObjetoGeográfico	Polígono
8, 18, 28	CampoGeográfico	PontosIrregulares
31 com partes 0 ou 1	CampoGeográfico	TIN
31 com partes 2, 3, 4, ou 5	ObjetoGeográfico	ObjetoComplexo com agregação de Polígono

Os limites de abrangência dos eixos, determinados pelo item *Bounding Box* do cabeçalho do arquivo principal, podem ser mapeados como um metadado na classe GeoMetadado se o tipo de dado for uma subclasse de FenômenoGeográfico, ou da classe Metadado se for subclasse da classe ObjetoNãoGeográfico.

Através dos campos que compõem o arquivo de dados descritivos pode-se identificar atributos (seus tipos e tamanhos) do tipo de dado que está sendo importado.

3.3.2 Regras de Mapeamento da GML para GeoFrame

Para ser possível que as informações geográficas codificadas em um arquivo de formato GML sejam interpretadas corretamente, com o propósito da engenharia reversa, será considerado que o arquivo atenda aos seguintes requisitos:

- ✓ estar de acordo com as normas estabelecidas pela GML na versão 2.0;
- ✓ todos os arquivos de esquema devem estar disponíveis no mesmo caminho que o arquivo origem;
- ✓ só serão interpretados arquivos que tenham seu próprio esquema de aplicação; e
- ✓ todos as feições que estiverem descritas no arquivo origem devem estar codificadas como substitutas de uma das seguintes classes abstratas: _Feature, _featureCollection, e _featureMember. Estas classes estão definidas no esquema de feições da GML.

Cada arquivo GML será considerado um pacote Tema do GeoFrame. Inicia-se o processo de mapeamento identificando se o arquivo XML possui referência a um

esquema de aplicação próprio. Se existir, identifica-se todos os tipos de feição que estão declarados do arquivo de esquema, associado-as a uma das classes abstratas: _Feature, _featureCollection, ou featureMember. A associação é feita através do atributo substitutionGroup pertencente à declaração de elementos na XML. Quando a feição não estiver associada diretamente, deve existir uma definição de tipo e este deverá estar associado a uma das classes abstratas.

Através do arquivo de esquema, ainda, é possível identificar todas as propriedades que pertencem a uma determinada feição. As declarações de tipos de feição, que definem as propriedades, são feitas através dos elementos <complextype> e <simpletype> da XML.

Todas as feições substitutas de _Feature ou _featureCollection são consideradas subclasses da classe ObjetoNãoGeográfico do GeoFrame, se não possuírem atributos geométricos, e subclasses de ObjetoGeográfico, se tiverem atributos geométricos. Nenhuma feição pode ser considerada subclasse de CampoGeográfico porque a GML 2.0 não fornece suporte para objetos na visão de campo. São consideradas propriedades geométricas aquelas cujo tipo substitui a classe abstrata, declarada no esquema de feição da GML, _geometryProperty ou a classe abstrata _Geometry, declarada no esquema geométrico da GML. As propriedades geométricas são mapeadas para as especializações da classe ObjetoEspacial do GeoFrame conforme a Tabela 3.8.

TABELA 3.8 - Mapeamento das propriedades geométricas da GML para especializações da classe ObjetoEspacial do GeoFrame.

Elemento GML	**Classe GeoFrame**	**Agregação com Especializações de ObjetoEspacial**
PointProperty	Ponto	-
PolygonProperty	Polígono	-
LineStringProperty	Linha	-
MultiPointProperty	ObjetoComplexo	Ponto
MultiPolygonProperty	ObjetoComplexo	Polígono
MultiLineStringProperty	ObjetoComplexo	Linha
MultiGeometryProperty	ObjetoComplexo	Ponto, Polígono e Linha
Point	Ponto	-
LineString	Linha	-
LinearRing	Polígono	-
Polygon	Polígono	-
Box	Polígono	-
MultiPoint	ObjetoComplexo	Ponto
MultiLineString	ObjetoComplexo	Linha
MultiPolygon	ObjetoComplexo	Polígono
MultiGeometry	ObjetoComplexo	Ponto, Polígono e Linha

É possível identificar três tipos de associações no mapeamento de GML para GeoFrame: generalização, agregação e associação simples. A generalização é identificada através dos atributos <extention base> ou <restriction base> na declaração dos elementos <complexType> e <simpleType> da XML. Estes atributos indicam a herança de classes. A agregação é identificada através das feições do tipo _featureMember que, juntamente com as instâncias no arquivo XML, permitem a identificação de associações de agregação. Esse tipo de associação ocorre sempre com

feições do tipo _featureCollection. E, finalmente, as associações simples que podem ser identificadas através de feições que tenham como propriedade ligações a outros elementos definidos no esquema da aplicação através de atributos do esquema Xlink da GML. A cardinalidade das associações é definida através dos atributos minOccurs e maxOccurs.

A GML mostrou um grande grau de compatibilidade com o GeoFrame. Dados armazenados neste formato poderão ser mapeados para um esquema em GeoFrame com pouca ou nenhuma perda de informação, desde que o arquivo origem seja compatível com as restrições definidas anteriormente.

3.3.3 Regras de Mapeamento do SAIF para GeoFrame

O SAIF tem um poder de expressão superior ao GeoFrame. Relacionamentos topológicos entre objetos e atributos temporais são alguns dos exemplos de informação que não podem ser representadas no GeoFrame e que o SAIF representa. Sendo assim, um modelo de dados descrito utilizando SAIF, ao ser representado em um esquema GeoFrame pode perder informações, dependendo dos dados que estejam sendo representados. Entretanto, as informações que são consideradas essenciais no modelo, ou seja, as representações dos fenômenos do mundo real podem ser representadas no GeoFrame, assim como sua geometria.

O mapeamento é feito através do arquivo em notação CSN, o qual possibilita a identificação de todas as classes que compõem o esquema de dados dos fenômenos do mundo real que estão sendo processados.

Apenas serão consideradas classes que sejam especializações de GeographicObject. As demais serão desconsideradas pois não podem ser representadas no esquema GeoFrame. Especializações da classe GeographicObject que não tenham geometria são consideradas subclasses de ObjetoNãoGeográfico do GeoFrame, enquanto as demais serão consideradas subclasses de FenômenoGeográfico e suas especializações. Neste caso, dependendo do tipo de geometria que a classe represente será considerada subclassse de ObjetoGeográfico ou CampoGeográfico.

A classe SpatialDataSet representa um conjunto de objetos GeographicObject que são agrupados por questões de similaridade. No GeoFrame esses agrupamentos são representados através de pacotes. Objetos geográficos que sejam agregações de SpatialDataSet serão mapeados no GeoFrame em pacotes distintos. O mapeamento das especializações de GeographicObject serão representadas no GeoFrame conforme a Tabela 3.9.

Se a classe definida em CSN/SAIF não possuir um correspondente na tabela acima, sua superclasse é analisada. Se não possuir uma generalização, então a classe é ignorada, pois não faz parte dos tipos capazes de serem representados no GeoFrame. Se a superclasse também não tiver correspondência, analisa-se sua superclasse, e assim sucessivamente até chegar em uma classe que possua representação no GeoFrame.

Os relacionamentos entre as classes podem ser identificados de três maneiras. A primeira delas é a generalização, que é representada através das cláusulas superclass e subclass informadas na definição de cada classe. Agregações e associações simples são definidas na classe através da cláusula attributes. Para que seja considerado uma associação, o tipo do atributo deve ser uma classe definida no Esquema do Usuário ou

no Esquema Padrão do SAIF. Se o atributo possuir a cláusula set, indica uma agregação, caso contrário, é considerado uma associação simples.

TABELA 3.9 - Mapeamento do SAIF para GeoFrame.

Classe SAIF	**Classe GeoFrame**	**Agregação com Especializações de ObjetoEspacial ou RepresentaçãoCampo**
CellularStructure	CampoGeográfico	GradeCélula
Field	CampoGeográfico	TIN
IrregularDEM	CampoGeográfico	PontosIrregulares
Isolines	CampoGeográfico	Isolinhas
PointGrid	CampoGeográfico	GradePontos
Raster	CampoGeográfico	GradeCélula
TIN	CampoGeográfico	TIN
SpatialReferencing	FenômenoGeográfico	GeoMetadado
AnnotatedSpatialDataSet	ObjetoGeográfico	Ponto
GeometricAggregate	ObjetoGeográfico	ObjetoComplexo
Graph	ObjetoGeográfico	ObjetoComplexo
Point	ObjetoGeográfico	Ponto
VectorArea	ObjetoGeográfico	Polígono
VectorLine	ObjetoGeográfico	Linha
VectorVolume	ObjetoGeográfico	Polígono

4 Módulo Tradutor

O módulo Tradutor, componente da arquitetura descrita no Capítulo 2, é detalhado neste Capítulo. Este módulo é responsável pela aplicação das regras de mapeamento definidas no Capítulo 3. Por solicitação do módulo Engenharia Reversa, o Tradutor traduz o conteúdo de um arquivo de formato aceitável para um conjunto de tipos de dado e atributos. Dependendo do formato de entrada, um segundo conjunto pode ser fornecido, este com os relacionamentos entre os novos tipos de dado.

O diagrama de classes deste módulo, apresentado na Figura 4.1, é composto pela classe Tradutor e suas especializações: TradutorGML, TradutorShapefile, e TradutorSAIF. Além destas, também existe a classe Elementos, utilizada na tradução de arquivos no formato GML.

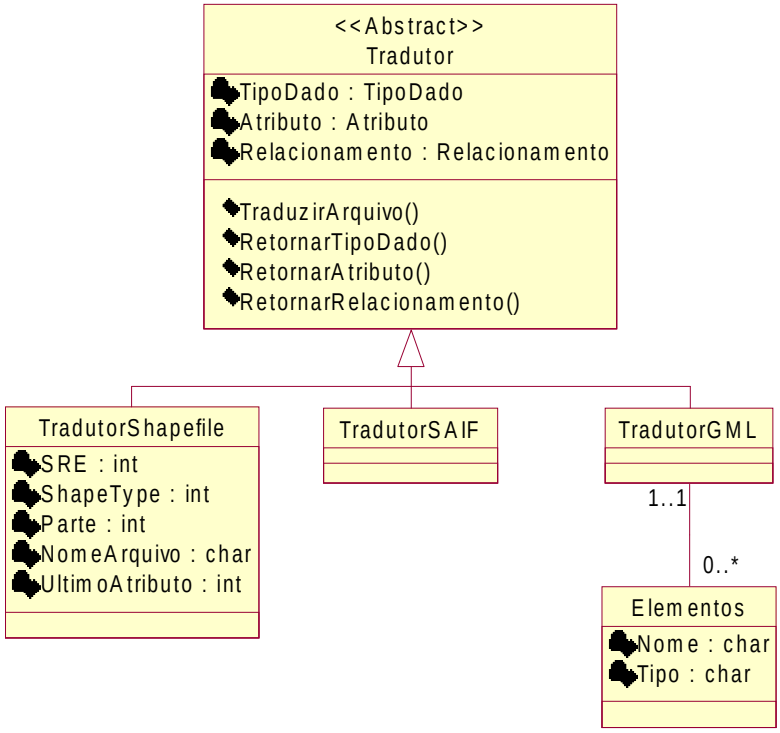

FIGURA 4.1 – Diagrama de classes do pacote Tradutor.

A classe Tradutor foi definida no Capítulo 2. Esta é uma classe abstrata, responsável pela extração dos tipos de dados do arquivo de entrada, seus atributos e relacionamentos.

A seguir serão definidas as especializações da classe Tradutor, sendo que cada uma delas é responsável pela tradução de um formato de dados geográficos aceito pelo sistema de engenharia reversa. Para que um novo formato de transferência seja aceito pelo sistema de engenharia reversa, uma nova especialização do Tradutor deve ser definida.

4.1.1.1 Classe TradutorShapefile

A classe TradutorShapefile é responsável pela extração de tipos de dado a partir de arquivos de entrada com o formato Shapefile discutido na Seção 3.1.1. O atributo SRE indica o sistema de referência espacial em que os tipos de dado do arquivo estão armazenados. ShapeType indica o tipo de geometria dos tipos de dado contidos no arquivo. Parte armazena o tipo da parte quando a geometria for um objeto complexo. UltimoAtributo guarda o número do último atributo retornado. Esses atributos são obtidos através de informações gravadas no arquivo de entrada.

As atividades do método TraduzirArquivo() são ilustradas, na Figura 4.2, através de um diagrama de atividades e descritas a seguir:

a) Abre o arquivo principal fornecido pelo usuário. Verifica se os quatro primeiros bytes do arquivo têm o valor 9994, indicando que é um arquivo válido Shapefile. Caso o valor não seja encontrado ou qualquer outro erro ocorra, é envidada uma mensagem de erro ao usuário e finalizado o processo;

b) Extrai as seguintes informações do registro de cabeçalho do arquivo principal:

b.1) O sistema de referência espacial é lido nos bytes 36 a 92. Esta informação é armazenada no atributo SRE;

b.2) A geometria dos tipos de dado é lido nos bytes 32 a 35. Esta informação é armazenada no atributo ShapeType;

c) Armazena o nome do arquivo de entrada, passado como parâmetro, no atribudo NomeArquivo.

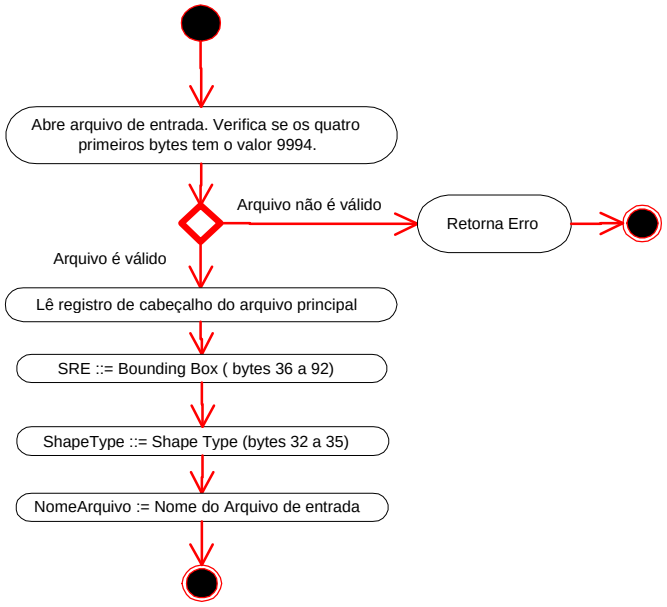

FIGURA 4.2 – Diagrama de atividades do método TraduzirArquivo da classe TradutorShapefile.

O método RetornarTipoDado(), da classe TradutorShapeFile, busca por um tipo de dado no arquivo de entrada, instancia o atributo TipoDado e atribui a ele o conteúdo do tipo de dado extraído do arquivo de entradas. Este atributo é o valor de retorno deste método. A Figura 4.3 apresenta o diagrama de atividades do método. Tais atividades são descritas a seguir:

a) Verifica o valor do atributo ShapeType. Se este for igual a 99, significa que não existem mais tipos de dado a serem extraídos do arquivo de entrada. Neste caso, retorna uma mensagem indicando o fim do processo. Caso contrário, realiza os próximos passos;

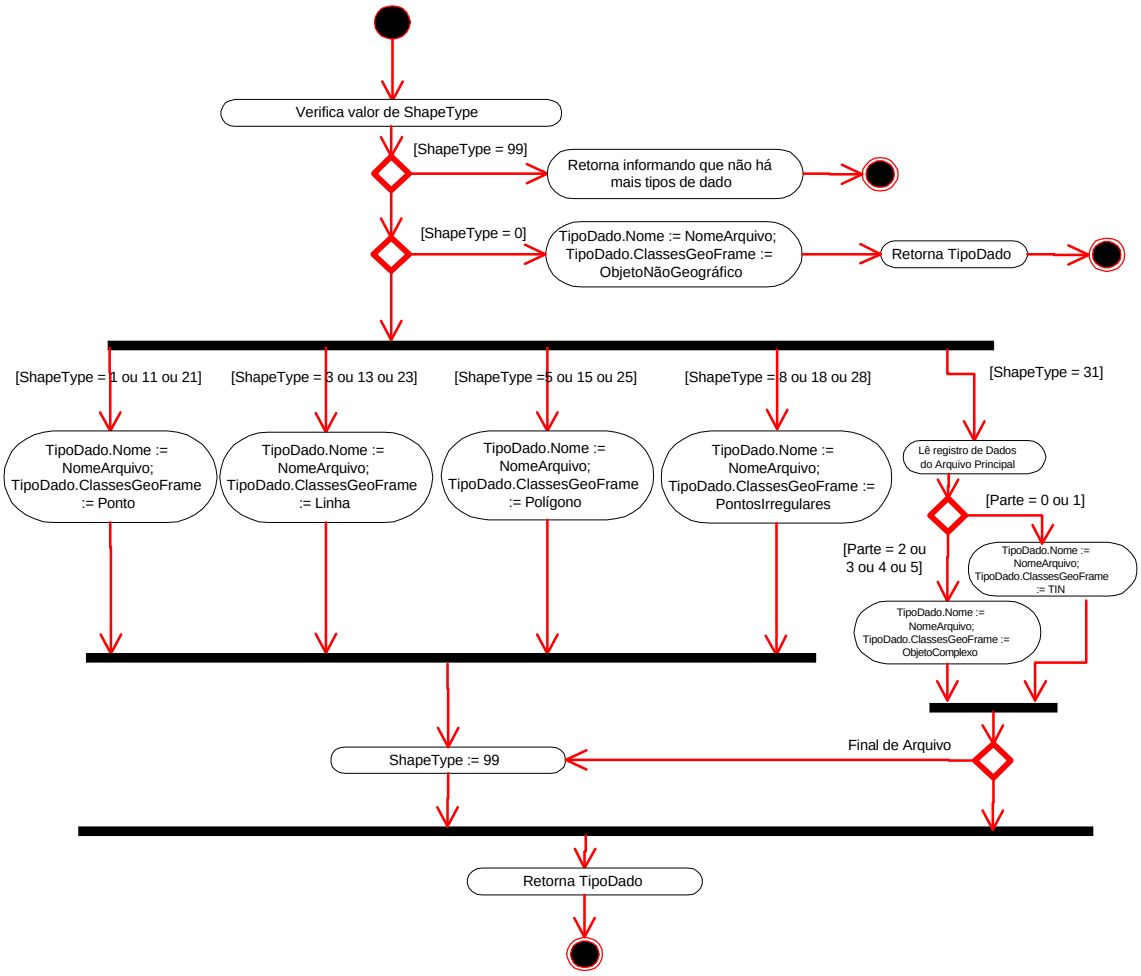

FIGURA 4.3 – Diagrama de atividades do método RetornarTipoDado da classe TradutorShapefile.

b) TipoDado é instanciado. Seu atributo Nome recebe o valor do atributo NomeArquivo da classe TradutorShapefile. Dependendo do tipo de geometria indicado pelo valor de ShapeType, o atributo ClasseGeoFrame recebe o valor de um tipo de classe do *framework* GeoFrame:

b.1) ShapeType igual a 0, o atributo ClasseGeoFrame recebe o valor "ObjetoNãoGeográfico";

b.2) ShapeType igual a 1, 11 ou 21, o atributo ClasseGeoFrame recebe o valor "Ponto";

b.3) ShapeType igual a 3, 13 ou 23, o atributo ClasseGeoFrame recebe o valor "Linha";

b.4) ShapeType igual a 5, 15 ou 25, o atributo ClasseGeoFrame recebe o valor "Polígono";

b.5) ShapeType igual a 8, 18 ou 28, o atributo ClasseGeoFrame recebe o valor “PontosIrregulares”;

b.6) Se ShapeType igual a 31, é um objeto complexo e suas partes devem ser analisadas:

b.6.1) Se as partes forem 0 ou 1, o atributo ClasseGeoFrame recebe o valor “TIN”;

b.6.2) Se as partes forem 2, 3, 4, ou 5, o atributo ClasseGeoFrame recebe o valor “ObjetoComplexo Polígono”;

d) O atributo Shapetype recebe 99. Se ShapeType for igual a 31 (objeto complexo), só irá receber valor 99 se for final de arquivo;

e) Retorna o atributo TipoDado.

O método RetornarAtributo() retorna, a cada chamada, um novo atributo para o tipo de dado que é passado como parâmetro. O valor de retorno é indicado no atributo Atributo. Neste formato de entrada, apenas um tipo de dado é extraído para cada arquivo processado pelo sistema. Portanto, todos os atributos existentes no arquivo de informações descritivas do Shapefile são associados a esse tipo de dado. Os atributos são extraídos a partir do arquivo de dados descritivos no formato dBase. A Figura 4.4 mostra as atividades realizadas por este método.

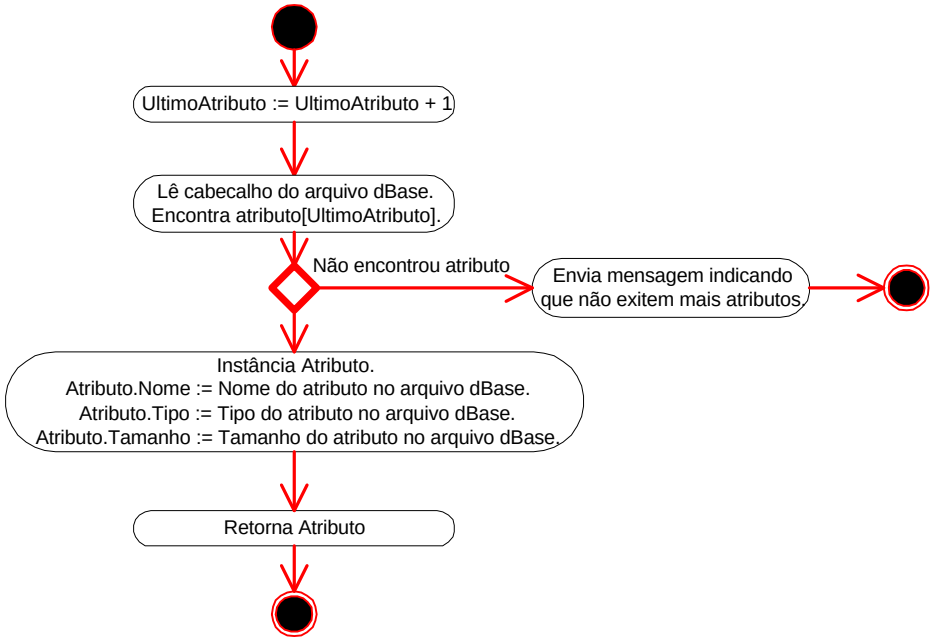

FIGURA 4.4 – Diagrama de atividades do método RetornarAtributo da classe TradutorShapefile.

O método RetornarRelacionamento() não é implementado na classe TradutorShapefile. Este formato de arquivo não possibilita a extração de informações de relacionamento entre tipos de dado, como foi descrito na Seção 3.1.1. Quando este método é invocado, apenas uma mensagem de que não existe relacionamentos é retornada.

4.1.1.2 Classe TradutorGML

A classe TradutorGML é responsável pela extração dos tipos de dado em arquivos de entrada com o formato GML, conforme a Seção 3.1.2. Para que possam ser extraídas informações de feições, seus atributos e relacionamentos, de arquivos neste formato, dois arquivos são analisados. Um primeiro arquivo em formato XML, contém as instâncias de feições, e um segundo arquivo em formato XML Schema, define o esquema da aplicação.

A classe TradutorGML está associada à classe Elementos. Nesta classe são instanciados objetos que representam as feições definidas no esquema da aplicação, as quais serão processadas pela classe TradutorGML. A seguir são descritos os métodos TraduzirArquivo(), RetornarTipoDado(), RetornarRelacionamento(), e RetornarAtributo() definidos na classe.

O método TraduzirArquivo() prepara as informações necessárias para que a tradução de um arquivo no formato GML possa ser feita. Recebe, como parâmetro, o nome de um arquivo XML. As atividades deste método são detalhadas a seguir e ilustradas na Figura 4.5:

a) O arquivo XML é aberto e verifica-se a existência de um esquema da aplicação. Se este não existir, a engenharia reversa não pode ser feita. Uma mensagem de erro é enviada ao usuário e o processo é encerrado. Caso contrário, o arquivo XSD definido é aberto;

b) Os *namespaces* dos arquivos XML e XSD são tratados. Se existir alguma inconsistência na definição destes, o processo de engenharia reversa é finalizado;

c) Para cada elemento identificado no esquema da aplicação faz-se:

c.1) Verifica valor do atributo substitutionGroup:

c.1.1) Se atributo substitutionGroup for igual Feature, _featureCollection, ou _featureMember, então instancia um objeto da classe Elementos.

c.1.2) Se SubstitutionGroup tem o valor de um tipo definido na aplicação, busca pela definição deste tipo e volta para passo (c.1);

c.1.3) Qualquer outro valor que SubstitutionGroup tenha, uma mensagem de erro será retornada ao usuário e o processo será finalizado.

Uma das restrições para que um arquivo no formato GML possa ser processado, conforme definição na Seção 3.1.2, é de que todas as feições que estiverem descritas no arquivo origem devem estar codificadas como substitutas de uma das três classes abstratas tratadas acima. Caso contrário o arquivo não será considerado válido.

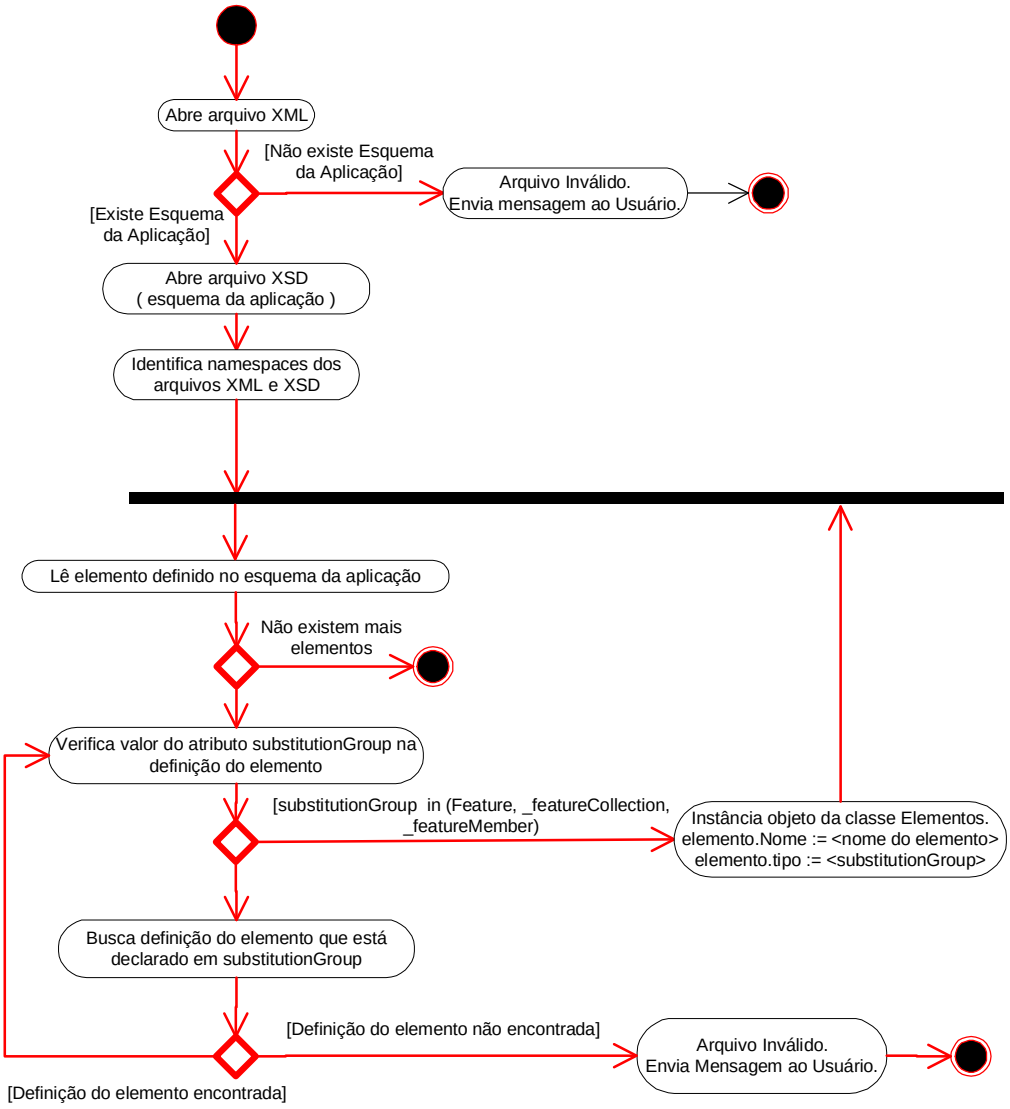

FIGURA 4.5 – Diagrama de Atividades do método TraduzirArquivo da classe TradutorGML.

O método RetornarTipoDado(), da classe TradutorGML, busca por um tipo de dado na lista de elementos preparada pelo método TraduzirArquivo(). Os tipos de dado mantidos na classe Elementos estão associados à classe TradutorGML. A cada chamada deste método, é retornado um tipo de dado que foi extraído de um arquivo em formato GML.

As atividades do métodos estão ilustradas na Figura 4.6 e descritas a seguir:

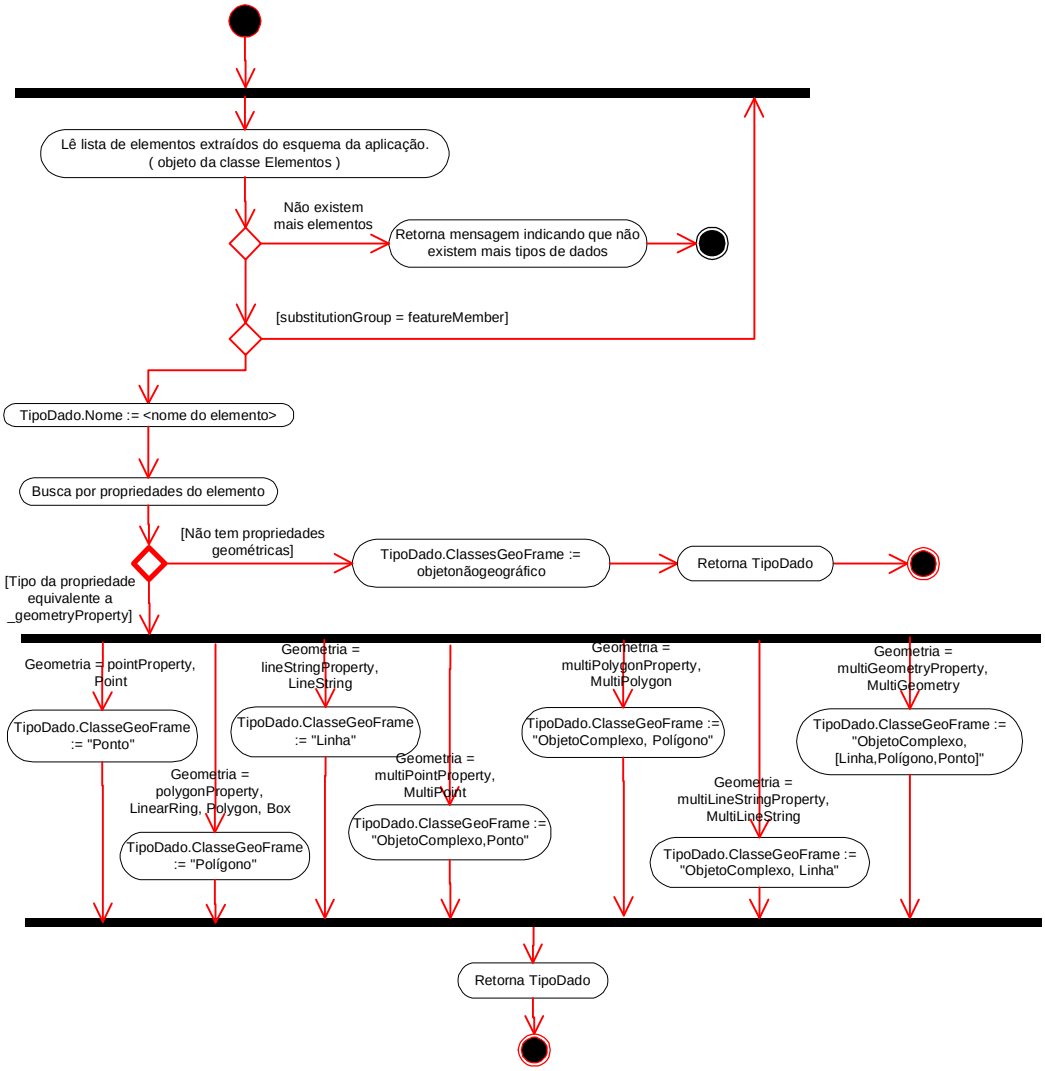

FIGURA 4.6 – Diagrama de Atividades do método RetornarTipoDado da classe TradutorGML.

a) Lê um elemento na lista de Elementos. Caso não existam mais elementos, retorna uma mensagem indicando o fim do processo;

b) Se o elemento lido for do tipo _featureMember, trata-se de um elemento de ligação e não de uma feição. Esses elementos não são tipos de dado, portanto, nestes casos volta-se para o passo (a);

c) TipoDado é instanciado. O atributo Nome recebe o nome do elemento da lista;

d) Busca todas as propriedades do elemento no arquivo XML. O atributos ClasseGeoFrame do objeto TipoDado recebe um dos seguintes valores:

d.1) Se nenhuma das propriedades for equivalente ao elemento _geometryProperty da GML, o objeto não é geográfico. Neste caso ClasseGeoFrame recebe o valor “ObjetoNãoGeográfico”;

d.2) Se propriedade é do tipo pointProperty ou Point, ClasseGeoFrame recebe o valor “Ponto”;

d.3) Se propriedade é do tipo polygonProperty, linearRing, Polygon, ou Box, ClasseGeoFrame recebe o valor “Polígono”;

d.4) Se propriedade é do tipo lineStringProperty ou LineString, ClasseGeoFrame recebe o valor “Linha”;

d.5) Se propriedade é do tipo multiPointProperty ou MultiPoint, ClasseGeoFrame recebe o valor “ObjetoComplexo Ponto”;

d.6) Se propriedade é do tipo multiPolygonProperty ou MultiPolygon, ClasseGeoFrame recebe o valor “ObjetoComplexo Polígono”;

d.7) Se propriedade é do tipo multilineStringProperty ou MultiLineString, ClasseGeoFrame recebe o valor “ObjetoComplexo Linha”;

d.8) Se propriedade é do tipo multiGeometryProperty ou MultiGeometry, ClasseGeoFrame recebe o valor “ObjetoComplexo Ponto, Linha, Polígono”;

e) Retorna TipoDado.

O método RetornarAtributo() busca pelos atributos do tipo de dado que foi passado como parâmetro. Retorna um atributo a cada vez que for chamado o método. Os atributos são extraídos a partir do arquivo XML. Cada elemento que esteja contido dentro da definição do tipo de dado, é considerado um atributo deste.

A Figura 4.7, ilustra o driagrama de atividades do método.

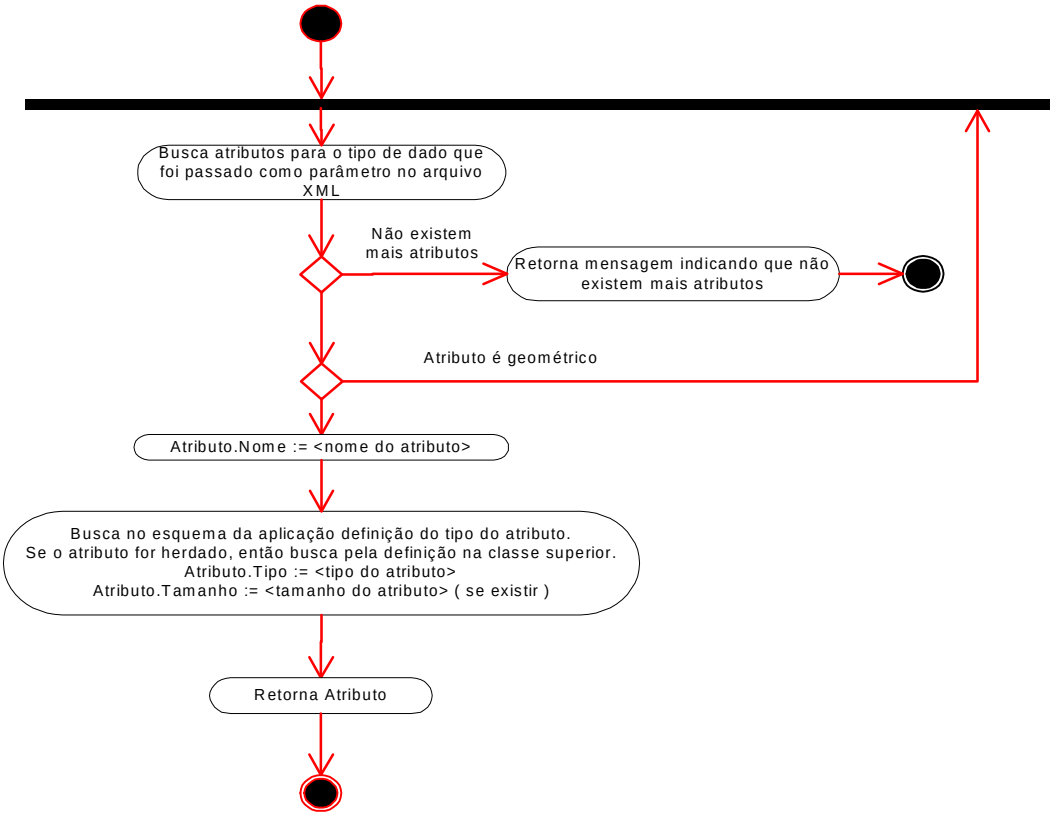

FIGURA 4.7 – Diagrama de Atividades do método RetornarAtributo da classe TraduzirGML.

As seguintes atividades são executadas pelo método:

a) Busca por um elemento atributo dentro da definição do tipo de dado. Se não existirem mais atributos, envia mensagem, notificando que não existem mais atributos para serem retornados;

b) Se o tipo do atributo for geométrico, equivalente a _geometryProperty, não é considerado um atributo. Atributos geométricos são utilizados para identificar o tipo de feição que está sendo processada. Neste caso, volta-se para o passo (a);

c) Instancia Atributo, e atribui o nome da propriedade ao atributo Nome deste objeto;

d) Busca no esquema da aplicação pelo tipo do atributo. Esta definição pode estar na definição do tipo de feição, no esquema da aplicação, ou ser um atributo definido no Esquema de Feições da GML. Atribui os valores encontrados aos atributos Tipo e Tamanho, respectivamente;

e) Retorna Atributo.

O método RetornarRelacionamento() é responsável por retornar os relacionamentos que existam para um determinado tipo de dado passado como parâmetro.

As atividades deste método são ilustradas em quatro diagramas que devem ser implementados, em seqüência, pelo método. O primeiro diagrama, ilustrado na Figura 4.8, trata de associações de generalização. Seu fluxo de atividades é descrito abaixo.

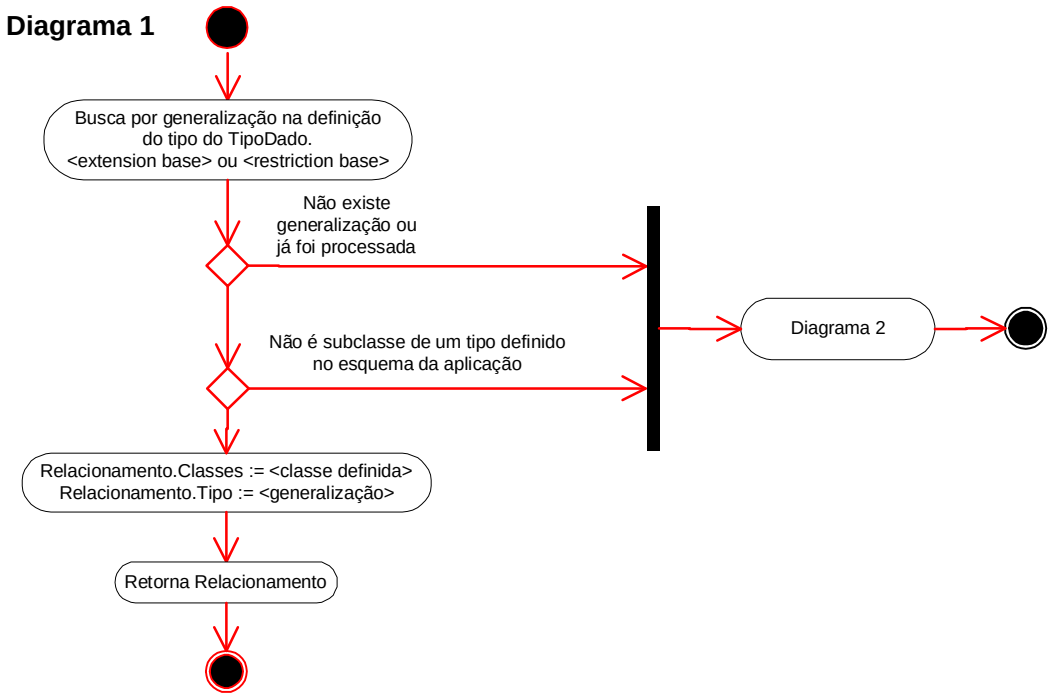

FIGURA 4.8 – Diagrama de atividades 1 do método RetornarRelacionamento da classe TraduzirGML.

a) Busca por generalizações na definição do tipo de dado. A generalização é definida através dos rótulos <extension base> ou <restriction base>. Caso não exista generalização, ou esta já tenha sido processada, passa para o diagrama 2;

b) Verifica se a classe indicada na generalização é um tipo definido no esquema da aplicação. Caso não seja, indica que não existe uma generalização entre feições da aplicação. Neste caso, passa para o diagrama 2;

c) Instância Relacionamento. O atributo Tipo recebe o valor “generalização”. O nome da classe definida no rótulo que define a generalização é atribuído ao atributo Classes;

d) Retorna Relacionamento.

As associações simples podem ser identificadas em dois casos que são tratados nos diagramas 2 e 3, respectivamente. No primeiro deles, a associação está definida dentro da definição do tipo do tipo de dado através de um atributo de ligação Xlink. Um exemplo de definição em GML para este caso seria:

```
<complexType name="MunicipioType">
    <complexContent>
     <extension base="gml:AbstractFeatureType">
      <sequence>
          <element name="nome" type="string"/>
          <element name="populacao" type="integer"/>
          <element ref="gml:extentOf"/>
          <element ref="ex:empresa" minOccurs="0"/>
      </sequence>
     </extension>
    </complexContent>
</complexType>
```

No exemplo, acima, é definida uma associação do tipo MunicipioType com um tipo de dado chamado empresa. O atributo minOccurs indica que a cardinalidade mínima do relacionamento é zero. A cardinalidade máxima é indefinida, uma vez que o atributo maxOccurs foi omitido. O diagrama 2 do método RetornarRelacionamento(), ilustrado na Figura 4.9, descreve as atividades executadas para identificar associações deste tipo. A seguir são descritas suas atividades:

a) Busca por associações simples na definição dos tipos de elementos. Se não existir associações desse tipo, ou se estas já tiverem sido processadas, passa para o diagrama 3;

b) Encontra as cardinalidades mínima e máxima do relacionamento através dos atributos minOccurs e maxOccurs, respectivamente. Se maxOccurs for omitido, indica que a cardinalidade máxima é indefinida;

c) Instancia Relacionamento. Os atributos de Relacionamento recebem os seguintes valores: Classes recebe o nome do elemento que estiver associado no atributo Xlink; Tipo recebe o valor “associação simple”; Card1min recebe zero, se o atributo não for obrigatório, e um se for obrigatório; Card1Max recebe um; Card2Min recebe o valor do atributo minOccurs; e Card2Max recebe o valor do atributo maxOccurs;

d) Retorna Relacionamento.

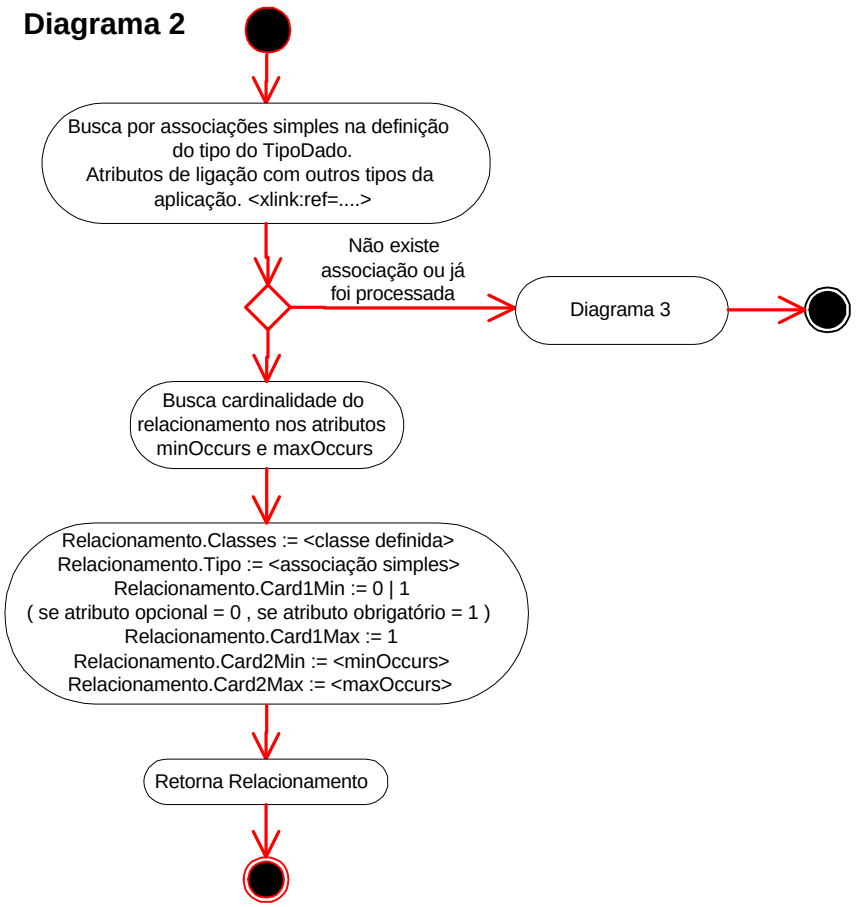

FIGURA 4.9 – Diagrama de atividades 2 do método RetornarRelacionamento da classe TraduzirGML.

No segundo caso, onde podem ser encontradas associações simples, um tipo de elemento específico para associação pode ser definido. O exemplo, a seguir, ilustra esta situação.

```
<complexType name="municipioEmpresaType">
    <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
           <element name="producao" type="string"/>
           <element ref="ex:municipio" minOccurs="1" maxOccurs="1" />
           <element ref="ex:empresa" minOccurs="1" maxOccurs="1" />
        </sequence>
      </extension>
    </complexContent>
</complexType>
```

No exemplo acima, um tipo de elemento chamado municipioEmpresaType foi definido, indicando que os elementos municipio e empresa se relacionam. A Figura 4.10 apresenta o diagrama 3 do método RetornarRelacionamento(). Este diagrama executa as seguintes atividades:

a) Busca por definições de tipo que sejam exclusivas para associações e que contenham o tipo de dado passado como parâmetro. Para que uma definição de tipo seja

considerada exclusiva para associações, esta deve conter apenas atributos de ligação a elementos definidos no esquema da aplicação;

b) Se não existir definições de tipo que atendam aos requisitos, ou se as mesmos já tiverem sido processadas, passa para o diagrama 4;

c) Instância Relacionamento. O atributo Tipo recebe o valor “associação simples”;

d) Busca pelo atributo de referência ao tipo de dado passado como parâmetro. Atribui o valor dos atributos minOccurs e maxOccurs para Card1Min e Card1Max, respectivamente;

e) Busca pelo atributo de referência do outro elemento que está sendo associado. O atributo Classe recebe o nome deste elemento. Atribui o valor dos atributos minOccurs e maxOccurs para Card2Min e Card2Max, respectivamente;

f) Retorna Relacionamento.

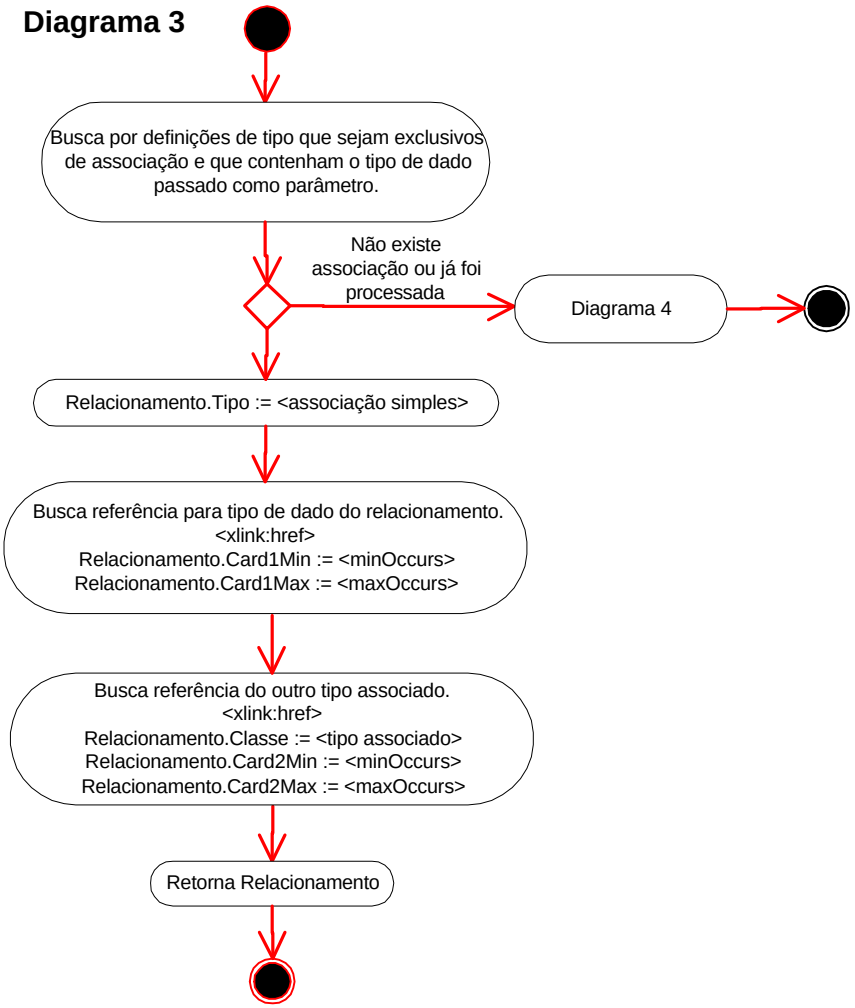

FIGURA 4.10 – Diagrama de atividades 3 do método RetornarRelacionamento da classe TraduzirGML.

O quarto diagrama do método RetornarRelacionamento diz respeito às associações de agregação. Este tipo de relacionamento precisa ser identificado no arquivo XML, analisando-se as ocorrências dos elementos definidos no esquema da

aplicação. O elemento que indica haver uma associação deste tipo é _featureMember.

No trecho em XML definido, abaixo, um elemento chamado estado possui agregação com um elemento chamado municipio. O elemento membroEstado é do tipo _featureMember, o que indica a agregação entre os outros dois elementos. Estado é do tipo _featureCollection e municipio é do tipo Feature.

```
<estado>
        <nome>Rio Grande do Sul</nome>
        <membroEstado>
                <municipio>
                        <nome>Porto Alegre</nome>
                        ...
                </municipio>
        </membroEstado>
        ...
        <membroEstado>
                <municipio>
                        <nome>Caxias do Sul</nome>
                        ...
                </municipio>
        </membroEstado>
</estado>
```

A Figura 4.11 ilustra as atividades deste diagrama.

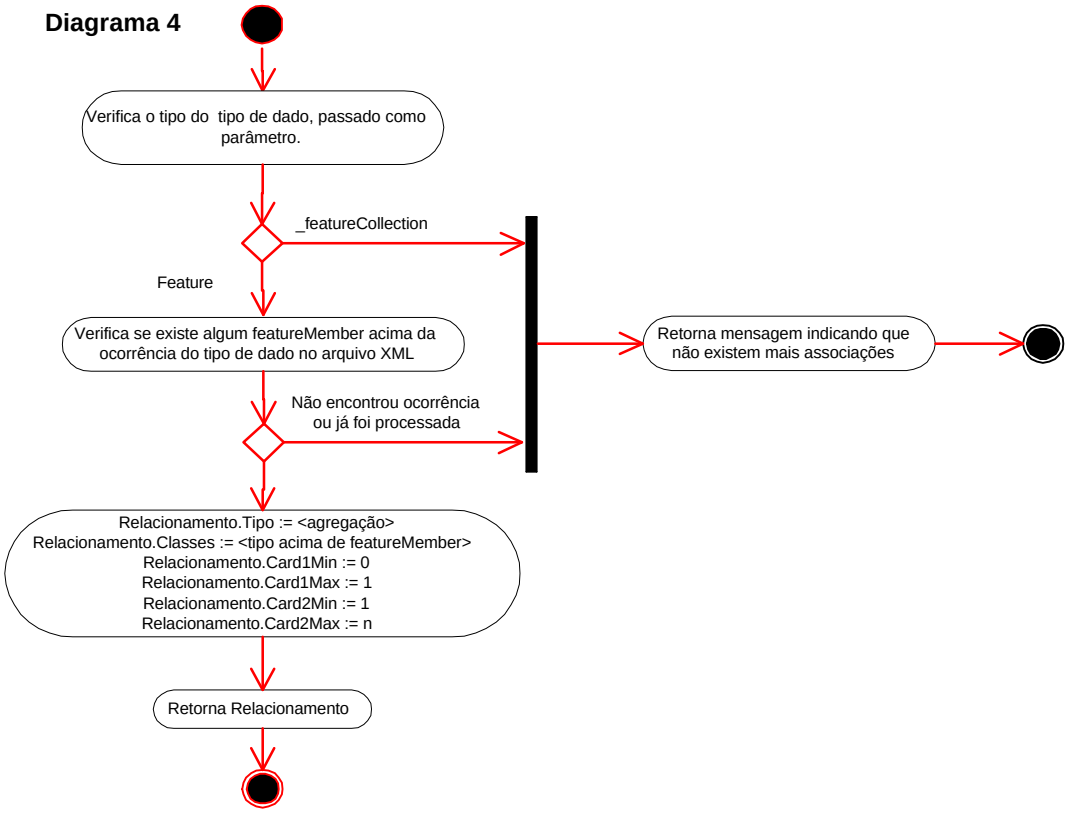

FIGURA 4.11 – Diagrama de atividades 4 do método RetornarRelacionamento da classe TraduzirGML.

O diagrama 4 executa as seguintes atividades:

a) Verifica se o tipo de dado passado como parâmetro é uma coleção de feições. Neste caso, envia uma mensagem ao usuário, indicando que não existem mais associações a serem retornadas. Quando o tipo de dado for uma coleção de feições, o elemento que estiver sendo associado a ele ainda não foi retornado. Portanto, não é necessário retornar a associação;

b) O tipo de dado é uma feição. Verifica a ocorrência de algum elemento _featureMember acima de sua ocorrência no arquivo XML. Ou seja, o tipo de dado deve estar contido em um _featureMember. Se não encontrar nenhuma ocorrência, retorna mensagem indicando que não existem mais associações a serem retornadas;

c) Instancia Relacionamento. Seus atributos recebem os seguintes valores: Tipo recebe o valor "agregação"; Classes recebe o nome do elemento em que _featureMember está contido; Card1Min rebece zero e Card1Max recebe um; Card2Min recebe valor um e Card2Max recebe "n".

4.1.1.3 Classe TradutorSAIF

A classe TradutorSAIF é responsável pela extração dos tipos de dados em arquivos de entrada com o formato SAIF/CSN, discutido na Seção 3.1.3. No arquivo em formato CSN está o esquema de dados definido pelo usuário. A seguir um exemplo de definição em CSN, para suportar o esquema GeoFrame definido na Figura 4.12.

```
<GeographicObject
        subclass:    Estado
        attributes:  PIB: Double
                     position.geometry:polygon
                     Regiões set(Região)
>
<GeographicObject
        subclass:    Região
        attributes:  nome: string
                     sede_regional: string
                     position.geometry:polygon
                     Municipios set(Municipio)
>
<GeographicObject
        subclass:    Municipio
        attributes:  nome: string
                     dt_ato_criacao: date
                     sede_municipio: string
                     referencia_legal_ato_criacao: string
                     referencia_alteracoes_legal: string
                     nro_eleitores: integer
                     position.geometry:polygon
                     [municipioOrigem] Municipio
                     Distritos set(Distrito)
                     Setores set(SetorCensitario)
>
<GeographicObject
        subclass:    Distrito
        attributes:  nome: string
                     position.geometry:polygon
```

Bairros set(Bairro)
>
<GeographicObject
subclass: SetorCensitario
attributes: cod_IBGE: string
position.geometry:polygon
>
<GeographicObject
subclass: Bairro
attributes: nome: string
position.geometry:polygon >

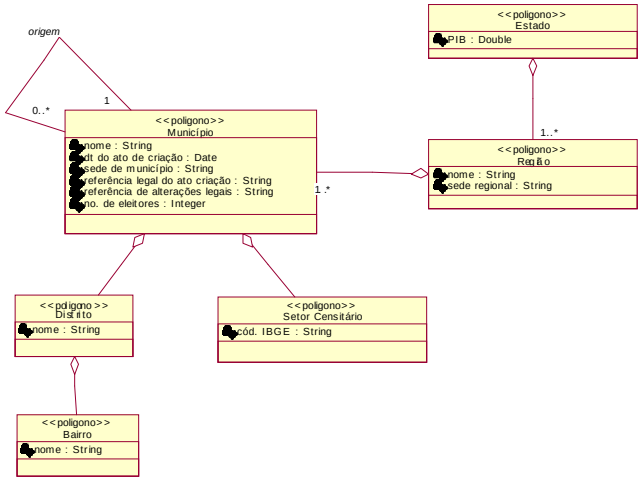

FIGURA 4.12 – Exemplo de esquema GeoFrame definido em CSN.

A seguir serão definidos os métodos implementados por esta classe, são eles: TraduzirArquivo, RetornarTipoDado, RetornarAtributo, e RetornarRelacionamento.

O método TraduzirArquivo() verifica se o arquivo indicado é válido. A Figura 4.13 ilustra o diagrama de atividades deste método.

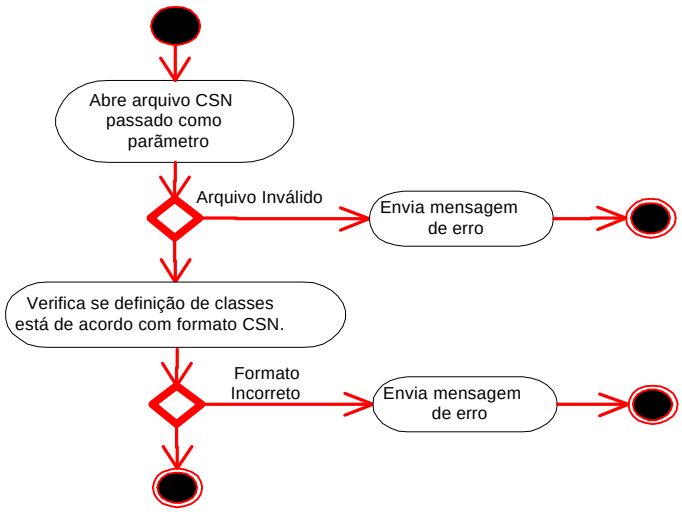

FIGURA 4.13 – Diagrama de atividades do método TraduzirArquivo da classe TraduzirSAIF.

Este método executa as seguintes atividades:

a) Abre arquivo informado como parâmetro. Se ocorrer qualquer erro na abertura retorna mensagem de arquivo inválido e finaliza o processo;

b) Verifica se a sintaxe de definição das classes está de acordo com o formato CSN. Se encontrar alguma inconsistência, envia mensagem indicando que o arquivo não é válido. A sintaxe de definição de classes no formato CSN foi discutido na Seção 3.1.3.4.

O método RetornarTipoDado() analisa um arquivo em formato CSN e retorna um novo tipo de dado a cada vez que é chamado. Cada definição de classe existente no arquivo indica um tipo de dado. O método executa as atividades a seguir, ilustradas na Figura 4.14.

a) Lê definição de uma classe no arquivo CSN. Se encontrar fim de arquivo, retorna mensagem indicando que o processo terminou. Não existem mais tipos de dado a serem retornados;

b) Se a classe definida na cláusula superclassing não fizer parte do esquema do usuário, nem do Esquema Padrão do SAIF, volta para o passo (a);

c) Se a classe definida na cláusula superclassing não for subclasse de GeographicObject, direta ou indiretamente, volta para o passo (a);

d) Instancia TipoDado. O atributo Nome recebe o valor da cláusula subclassing;

e) Busca pela definição de um atributo chamado position.geometry. Este atributo indica que a classe possui uma associação com a classe SpatialObject, ou seja, é um objeto geográfico com geometria. Classifica o tipo de dado como a seguir:

e.1) Não existe atributo position.geometry, neste caso TipoDado.ClasseGeoFrame recebe o valor “ObjetoNãoGeográfico”;

e.2) O valor de position.geometry é nulo, neste caso TipoDado. ClasseGeoFrame recebe o valor “ObjetoNãoGeográfico”;

e.3) O valor de position.geometry é igual a Point ou AnnotateadSpatialDataSet, TipoDado. ClasseGeoFrame recebe o valor “Ponto”;

e.4) O valor de position.geometry é igual a Field ou TIN, TipoDado. ClasseGeoFrame recebe o valor “TIN”;

e.5) O valor de position.geometry é igual a IrregularDEM, TipoDado.ClasseGeoFrame recebe o valor “PontosIrregulares”;

e.6) O valor de position.geometry é igual a Raster ou CellularStructure, TipoDado.ClasseGeoFrame recebe o valor “GradeCélulas”;

e.7) O valor de position.geometry é VectorLine, neste caso TipoDado.ClasseGeoFrame recebe o valor “Linha”;

e.8) O valor de position.geometry é VectorArea ou VectorVolume, TipoDado.ClasseGeoFrame recebe o valor “Polígono”;

e.9) O valor de position.geometry é Isolines, neste caso TipoDado.ClasseGeoFrame recebe o valor “Isolinhas”;

e.10) O valor de position.geometry é PointGrid, neste caso TipoDado.ClasseGeoFrame recebe o valor “GradePontos”;

e.11) O valor de position.geometry é SpatialReferencing, neste caso TipoDado.ClasseGeoFrame recebe o valor “TIN”;

e.12) O valor não for igual a nenhuma das alternativas acima, volta para o passo (a);

f) Retorna TipoDado.

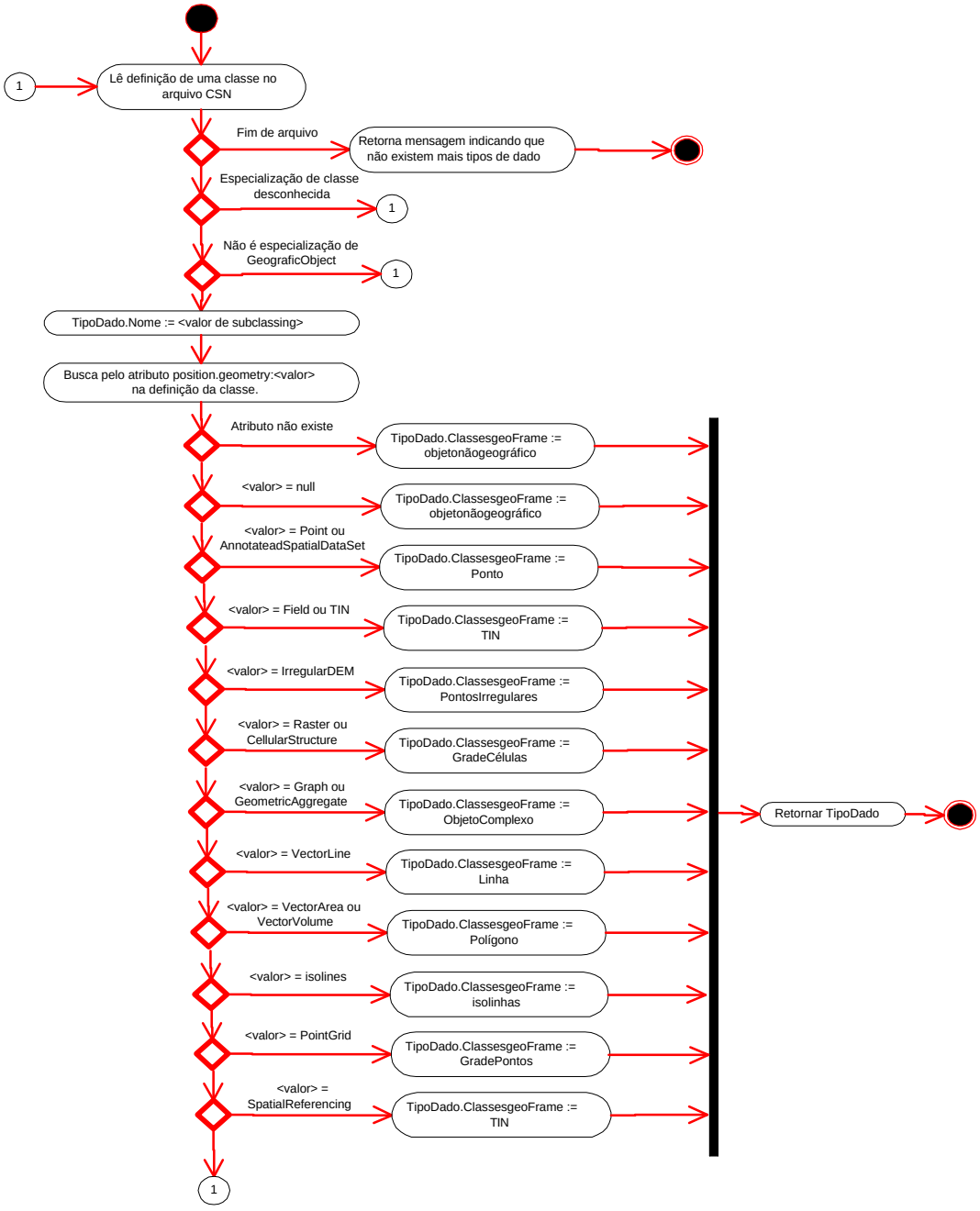

FIGURA 4.14 – Diagrama de atividades do método RetornarTipoDado da classe TraduzirSAIF.

O método RetornarAtributo() recebe, como parâmetro, o nome do tipo de dado que está sendo processado e retorna seus atributos, caso existirem. Para cada chamada deste método um atributo será retornado. Executa as atividades descritas a seguir e ilustradas na Figura 4.15, através de um diagrama de atividades.

a) Encontra a definição da classe cujo nome foi passado como parâmetro. Identifica cada um dos atributos definidos. Quando não houver mais atributos a serem retornados envia mensagem indicando fim do processo;

b) Verifica se o tipo do atributo é uma classe definida no esquema do usuário ou no modelo de dados do SAIF, caso seja volta para o passo (a). Atributos cujo tipo seja uma classe, indicam associações e serão tratados pelo método RetornarRelacionamento;

c) Instancia Atributo, suas propriedades recebem os seguintes valores: Nome, recebe o nome do atributo definido; Tipo, recebe o tipo do atributo definido; e Tamanho, recebe o tamanho do atributo definido caso existir;

d) Retorna Atributo.

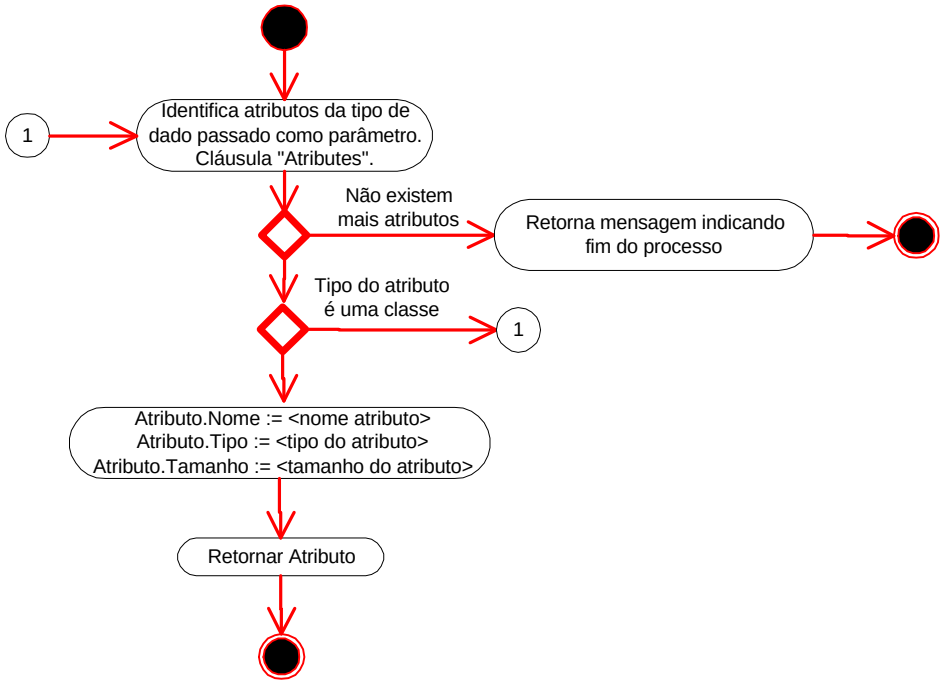

FIGURA 4.15 – Diagrama de atividades do método RetornarAtributo da classe TraduzirSAIF.

O método RetornarRelacionamento() busca na definição de classe, correspondente ao tipo de dado passado como parâmetro, seus relacionamentos a cada chamada do método. Executa as seguintes atividades, ilustrada no diagrama de classes da Figura 4.16:

a) Verifica se a generalização já foi retornada. Se sim, retorna mensagem informando que o processo foi finalizado;

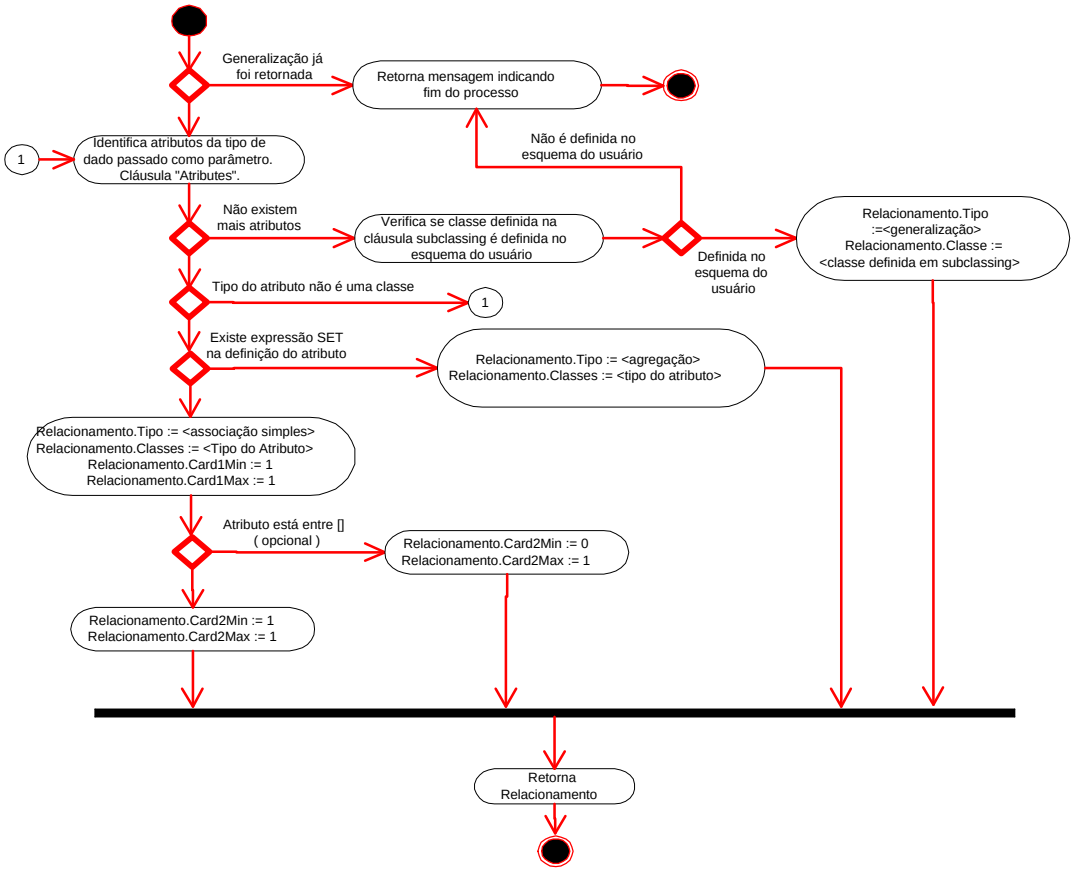

FIGURA 4.16 – Diagrama de atividades do método RetornarRelacionamento da classe TraduzirSAIF.

b) Identifica os atributos definidos na classe, através da cláusula attributes.

b.1) Se não exisistirem mais atributos, verifica se a classe definida na cláusula subclassing é do esquema do usuário. Se não for uma classe deste esquema retorna mensagem informando que o processo está finalizado;

b.1.1) Instancia Relacionamento. A propriedade Tipo recebe o valor “generalização”; A propriedade Classe recebe o nome da classe definida na cláusula subclassing;

b.1.2) Passa para o passo (e);

b.2) Se o tipo de atributo não for uma classe definida no esquema do usuário, ou no Esquema Padrão do SAIF, volta para passo (b);

b.3) Se existir a expressão SET na definição do atributo, indica que é uma agregação:

b.3.1) Instancia Relacionamento. A propriedade Tipo recebe o valor “agregação”; A propriedade Classe recebe o nome da classe definida como tipo do atributo;

b.3.2) Passa para o passo (e);

c) Instancia Relacionamento. A propriedade Tipo recebe o valor “associação simples”; Classes, receber o tipo do atributo; Card1Min e Card1Max, recebem o valor um;

d) Verifica se o nome do atributo está entre colchetes, indicando que é opcional.

d.1) Se for opcional as propriedades de Relacionamento recebem os seguintes valores: Card1Min recebe zero; e Card1Max recebe um. Passa para passo (e);

d.2) Não é opcional, as propriedades de Relacionamento recebem os seguintes valores: Card1Min recebe um; e Card1Max recebe um;

e) Retorna Relacionamento.

5 Estudo de Caso de Implementação

Para verificar a funcionalidade do Sistema de Engenharia Reversa porposto no Capítulo 2, desenvolveu-se um estudo de caso de implementação, o qual é apresentado a seguir.

O formato de transferência escolhido para o estudo de caso foi o Shapefile. Este é o formato, dentre os estudados, que disponibiliza menos informações para o sistema. Foi escolhido, no entanto, por ser, atualmente, o de maior utilização dentre os formatos estudados.

Cada arquivo, no formato shapefile, tem as informações de apenas um tipo de dado e nenhuma informação sobre relacionamentos, o que obriga o uso de padrões. Entende-se que, se for possível utilizar o sistema para este formato de transferência, será também possível usá-lo para os demais, uma vez que os outros formatos disponibilizam mais informações sobre os dados.

Mesmo que a definição de um Catálogo de Padrões não seja objetivo desse trabalho, para que fosse possível testar o sistema proposto, utilizando o formato Shapefile, foi necessário propor relacionamentos entre tipos de dado a partir de padrões. Para tal, são apresentados, no Anexo 1, padrões de análise extraídos de [LIS2000], além de padrões inferidos a partir do modelo de dados do Projeto SIME (Sistema de Informações Georreferenciadas da Região Metropolitana de Belém). Tais padrões devem ser consultados para que seja possível sugerir relacionamentos entre os tipos de dado que formarão o esquema conceitual final.

5.1 Formato e Conteúdo dos Arquivos de Entrada

Para avaliar a eficácia dos procedimentos de engenharia reversa propostos, foram utilizados dados e subesquemas do banco de dados do SIME.

O SIME, um projeto conjunto entre UFRGS e a Companhia de Habitação do Governo do Estado do Pará (COHAB/PA), tem como objetivo responder às demandas do governo e da sociedade civil, no tocante às informações sobre as características do próprio espaço e também àquelas relativas aos diversos processos que se desenvolvem no âmbito metropolitano de Belém. O SIME será implantado no Laboratório de Geoprocessamento da Diretoria de Assuntos Urbanos e Metropolitanos (DAU) da COHAB/PA que, juntamente com a Secretaria Executiva de Desenvolvimento Urbano e Regional (SEDURB), tem a atribuição de subsidiar as ações do Governo do Estado do Pará no tocante ao planejamento e à Gestão da Região Metropolitana de Belém.

Os dados do SIME foram disponibilizados através de arquivos, contendo dados geográficos da região de Belém no formato Shapefile. Estes arquivos foram utilizados como exemplos de entrada para o estudo de caso de implementação descrito abaixo.

5.2 Exemplo de Funcionamento do Sistema de Engenharia Reversa Proposto

Quando o sistema inicia, o método Iniciar() da classe Gerenciador_Engenharia_Reversa é invocado. Inicialmente, o método deve

solicitar, ao usuário, os parâmetros de entrada para o processo de engenharia reversa. Esta solicitação pode feita através de uma *interface* gráfica como a apresentada na Figura 5.1.

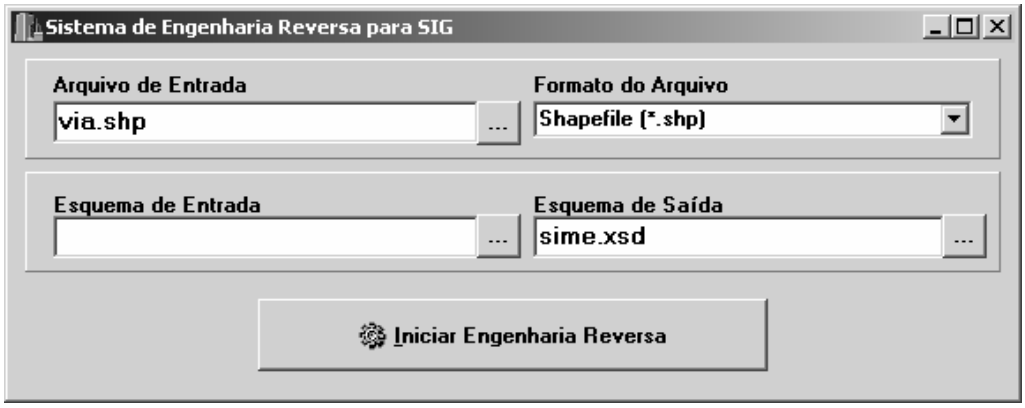

FIGURA 5.1 – *Interface* principal do protótipo.

Os parâmetros de entrada do processo de engenharia reversa são:

- ✓ Arquivo de Entrada: nome do arquivo de entrada para o qual o usuário deseja processar a engenharia reversa;
- ✓ Formato do Arquivo: formato do arquivo de entrada;
- ✓ Esquema de Entrada: nome de um arquivo que contenha um XML Schema do modelo de dados, já existente, a ser atualizado. Este parâmetro é opcional;
- ✓ Esquema de Saída: nome de um arquivo que conterá o XML Schema resultante do processo de engenharia reveresa.

Ao ativar o botão "Iniciar Engenharia Reversa", o método CriarEsquema() da classe Gerenciador_Esquema é invocado, passando, como parâmetro, o nome do arquivo informado em "Esquema de Saída". No exemplo acima, um arquivo chamado "sime.xsd" é criado, contendo a estrutura básica de um XML Schema como o da Figura 2.13. Nenhuma classe já existente será agregada ao esquema de saída uma vez que, neste exemplo, não foi informado um "Esquema de Entrada".

Após, o método Executar() da classe Gerenciador_Engenharia_Reversa é invocado. Todos os processos da engenharia reversa são iniciados através deste método já especificado no respectivo diagrama de atividades.

O primeiro arquivo que está sendo submetido é "via.shp". O sistema não encontra qualquer classe no esquema de dados. Portanto, o tipo de dado "via" é considerado novo e é agregado ao esquema de saída. Como não existem outros tipos de dado, nenhum relacionamento é sugerido. O trecho em XML Schema da Figura 5.2 deve ser agregado ao "Esquema de Saída".

```
<complexType name="viaType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <atribute name="codLogr" type="integer"/>
                        <atribute name="nomeLogr" type="string"/>
                </extension>
        </complexContent>
</complexType>
```

FIGURA 5.2 – Techo XML Schema para o tipo de dado "via".

Em um segundo momento, um novo arquivo é submetido ao sistema, através da *interface* principal, chamado “trechovia.shp”. É informado pelo usuário, ainda, um “Esquema de Entrada” chamado “sime.xsd” (criado pela execução anterior do sistema) e um “Esquema de Saída” chamado “sime2.xsd”.

Nesta segunda execução foi informado um “Esquema de Entrada” pelo usuário. Isso faz com que na criação do “Esquema de Saída” pelo método CriarEsquema() da classe Gerenciador_Esquema seja agregado, a este, os tipos de dado existentes no “Esquema de Entrada” informado.

No processo de engenharia reversa, o sistema não encontra um tipo de dado chamado “trechovia” no esquema existente, já que, até o momento, somente existe o tipo de dado “via”. O tipo de dado “trechovia” é agregado ao esquema de saída. Neste exemplo, foram utilizados os padrões 2 e 6, definidos no Anexo 1, para sugerir uma lista de associações possíveis para este novo tipo de dado. Uma *interface*, como a da Figura 5.3, é apresentada ao usuário.

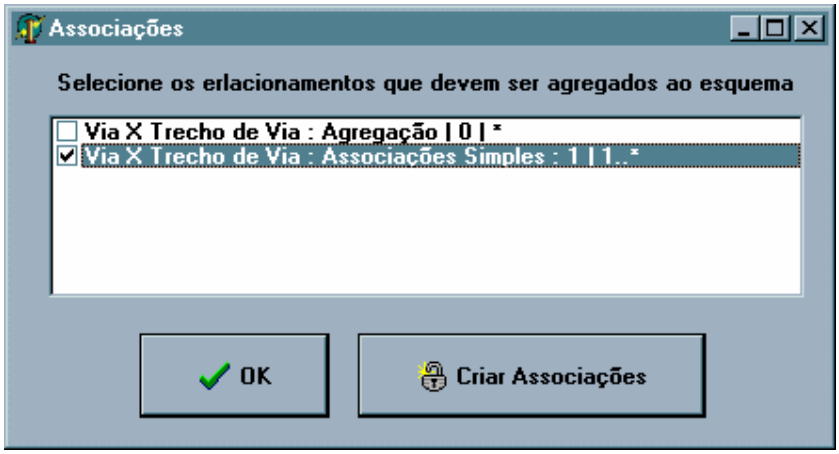

FIGURA 5.3 – *Interface* para escolher associações entre tipos de dados.

O botão “Criar Associações” deve ser utilizado quando o usuário deseja criar uma associação diferente das apresentadas. Ao ativar este botão será disponibilizado, ao usuário, a *interface* da Figura 5.6.

Para escolher uma das associações apresentadas, o usuário deve selecionar qual delas deseja adicionar ao esquema de saída e, então, ativar o botão “OK”. Neste exemplo será agregado, ao esquema, uma associação simples entre os tipos de dado “via” e “trechovia”. A Figura 5.4 ilustra o trecho em XML Schema que define o tipo de dado “TrechoVia” e sua associação com o tipo de dado “via”.

```
<complexType name="trechoviaType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <atribute name=”idVia” type=”integer”/>
                        <atribute name=”numInic” type=”integer”/>
                        <atribute name=”numFinal” type=”integer”/>
                </extension>
        </complexContent>
</complexType>

<complexType name="trechovia_viaType">
   <annotation>
        <appinfo>trechoviaId--trechoviaType.fid;viaId--viaType.fid </appinfo>
   </annotation>
```

```
   <complexContent>
        <restriction base="gml:FeatureAssociationType">
            <sequence>
               <element ref="hid:viaId"/>
               <element ref="hid:trechoviaId" minOccurs="0" maxOccurs="unbounded"/>
             </sequence>
             <attributeGroup ref="gml:AssociationAttributeGroup"/>
        </restriction>
    </complexContent>
</complexType>
```

FIGURA 5.4 – Trecho XML Schema para tipo de dado “TrechoVia”.

Em um terceiro momento, um novo arquivo, chamado “trechovia.shp”, é submetido ao sistema. É informado pelo usuário, ainda, um “Esquema de Entrada” chamado “sime2.xsd” (criado pela execução anterior do sistema) e um “Esquema de Saída” chamado “sime3.xsd”.

O tipo de dado do arquivo de entrada já faz parte do esquema da aplicação. Porém, existem novos atributos definidos no tipo de dado que está sendo submetido. Portanto, uma atualização é feita no esquema, agregando esses atributos. Na Figura 5.5 está ilustrado o trecho XML Schema que foi atualizado.

```
<complexType name="trechoviaType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <atribute name=”idVia” type=”integer”/>
                        <atribute name=”numInic” type=”integer”/>
                        <atribute name=”numFinal” type=”integer”/>
                          <atribute name=”larguramedia” type=”integer”/>
                          <atribute name=”tipopavimentacao” type=”char”/>
                          <atribute name=”sentido” type=”string”/>
                          <atribute name=”velocidademedia” type=”float”/>
                </extension>
        </complexContent>
</complexType>
```

FIGURA 5.5 – Trecho XML Schema para tipo de dado “TrechoVia” atualizado.

A seguir, o arquivo “trechocirculacao.shp” é submetido ao sistema. É informado pelo usuário, ainda, um “Esquema de Entrada” chamado “sime3.xsd” (criado pela execução anterior do sistema) e um “Esquema de Saída” chamado “sime4.xsd”.

Não é encontrado o tipo de dado da entrada no esquema atual da aplicação. Portanto, o novo tipo de dado (trechocirculacao) é agregado ao esquema da aplicação. O sistema não encontra nenhum padrão que contenha este tipo de dado. Desta forma, a *interface* que lista possíveis associações está vazia. O usuário tem, no entanto, a possibilidade de criar uma associação através da *interface* da Figura 5.6.

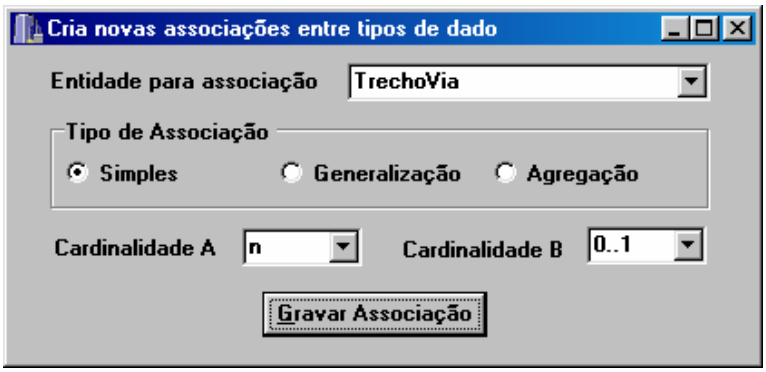

FIGURA 5.6 – *Interface* para criação de associações entre tipos de dado.

No exemplo da Figura 5.6, caso o usuário opte pela acão “Gravar Associação”, uma associação simples será inserida no esquema, entre os tipos de dado “TrechoCirculacao” e “TrechoVia”, com as cardinalidades selecionadas pelo usuário. A Figura 5.7 ilustra o trecho XML Schema que define o processo acima.

```
<complexType name="trechocirculacaoType">
        <complexContent>
                <extension base="gml:AbstractFeatureType"></extension>
        </complexContent>
</complexType>

<complexType name="trechovia_trechocirculacaoType">
   <annotation>
       <appinfo>trechoviaId--trechoviaType.fid;trechocirculacaoId--trechocirculacaoType.fid
</appinfo>
   </annotation>
   <complexContent>
        <restriction base="gml:FeatureAssociationType">
            <sequence>
               <element ref="hid:trechocirculacaoId" minOccurs="0" maxOccurs="1"/>
               <element ref="hid:trechoviaId" minOccurs="0" maxOccurs="unbounded"/>
             </sequence>
             <attributeGroup ref="gml:AssociationAttributeGroup"/>
        </restriction>
    </complexContent>
</complexType>
```

FIGURA 5.7 – Trecho XML Schema para tipo de dado “TrechoCirculacao”.

Ao final de cada execução do sistema um novo arquivo, contendo um XML Schema, é disponibilizado. No exemplo descrito, acima, os arquivos gerados como “Esquemas de Saída” foram utilizados como “Esquemas de Entrada” nas respectivas próximas execuções do protótipo. Depois das quatro execuções, um arquivo chamado “sime4.xsd” fica disponível, ao usuário, com todas as atualizações feitas. Na Figura 5.8, este arquivo é descrito.

```
<!-- ============================================================
      Cabeçalho XML Schema
 ============================================================ -->
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: sime4.xsd  -->
<schema targetNamespace="sime" mlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml=http://www.opengis.net/gml xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:hid="sime" elementFormDefault="qualified" version="2.03">
<annotation>
        <appinfo>sime4.xsd v2.03 2001-02</appinfo>
        <documentation xml:lang="en">
            GML schema for SIME
        </documentation>
</annotation>
<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/namespaces/gml/core/"
schemaLocation="http://www.opengis.net/namespaces/gml/core/feature.xsd"/>

<!-- ============================================================
      global element declarations
 ============================================================ -->
<element name="tema" type="hid:temaType"    substitutionGroup="gml:_FeatureCollection"/>
<element name="temaMember" type="hid:temaMemberType"
    substitutionGroup="gml:featureMember"/>
<element name="_temaFeature" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="gml:_Feature"/>
```

```
<element name="_via" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="rc:_temaFeature"/>
<element name="Via" type="rc:viaType" substitutionGroup="rc:_via"/>
<element name="_TrechoVia" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="rc:_temaFeature"/>
<element name=" _TrechoCirculacao" type="rc: TrechoViaType" substitutionGroup="rc:_
    TrechoVia"/>
<element name="_TrechoVia" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="rc:_temaFeature"/>
<element name="TrechoVia" type="rc: TrechoViaType" substitutionGroup="rc:_ TrechoVia"/>
<element name="TrechoVia _Via" type="rc:TrechoVia _ViaType"
    substitutionGroup="rc:_temaFeature"/>
<element name="_TrechoCirculacao" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="rc:_temaFeature"/>
<element name="TrechoCirculacao" type="rc:TrechoCirculacaoType"
    substitutionGroup="rc:_TrechoCirculacao "/>
<element name="TrechoVia _TrechoCirculacao" type="rc:TrechoVia _TrechoCirculacaoType"
    substitutionGroup="rc:_temaFeature"/>

<!-- ============================================================
      type definitions for region model
 ============================================================ -->
<complexType name="temaType">
        <complexContent>
                <extension base="gml:AbstractFeatureCollectionType">
                        <sequence>
                                <element name="name" type="string"/>
                                <element name="dateCreated" type="date"/>
                        </sequence>
                </extension>
        </complexContent>
</complexType>

<complexType name="temaMemberType">
        <complexContent>
                <restriction base="gml:FeatureAssociationType">
                        <sequence minOccurs="0">
                                <element ref="hid:_temaFeature"/>
                        </sequence>
                        <attributeGroup ref="gml:AssociationAttributeGroup"/>
                </restriction>
        </complexContent>
</complexType>

<!— ============================================================================
     Neste local estão definidas das classes que compõem o tema SIME
=============================================================================-->
<complexType name="viaType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <atribute name=”codLogr” type=”integer”/>
                        <atribute name=”nomeLogr” type=”string”/>
                </extension>
        </complexContent>
</complexType>

<complexType name="trechoviaType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <atribute name=”idVia” type=”integer”/>
                        <atribute name=”numInic” type=”integer”/>
                        <atribute name=”numFinal” type=”integer”/>
                        <atribute name=”larguramedia” type=”integer”/>
                        <atribute name=”tipopavimentacao” type=”char”/>
                        <atribute name=”sentido” type=”string”/>
```

```
                        <atribute name="velocidademedia" type="float"/>
                    </extension>
            </complexContent>
</complexType>

<complexType name="trechovia_viaType">
   <annotation>
        <appinfo>trechoviaId--trechoviaType.fid;viaId--viaType.fid </appinfo>
   </annotation>
   <complexContent>
          <restriction base="gml:FeatureAssociationType">
              <sequence>
                 <element ref="hid:viaId"/>
                 <element ref="hid:trechoviaId" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
               <attributeGroup ref="gml:AssociationAttributeGroup"/>
          </restriction>
    </complexContent>
</complexType>

<complexType name="trechocirculacaoType">
          <complexContent>
                    <extension base="gml:AbstractFeatureType"></extension>
          </complexContent>
</complexType>

<complexType name="trechovia_trechocirculacaoType">
   <annotation>
        <appinfo>trechoviaId--trechoviaType.fid;trechocirculacaoId--trechocirculacaoType.fid
</appinfo>
   </annotation>
   <complexContent>
          <restriction base="gml:FeatureAssociationType">
              <sequence>
                 <element ref="hid:trechocirculacaoId" minOccurs="0" maxOccurs="1"/>
                 <element ref="hid:trechoviaId" minOccurs="0" maxOccurs="unbounded"/>
               </sequence>
               <attributeGroup ref="gml:AssociationAttributeGroup"/>
          </restriction>
    </complexContent>
</complexType>

<!— ============================================================================
     Fim da definição das classes que compõem o tema SIME
=============================================================================-->

<!-- Id Property -->
<complexType name="idPropertyType">
          <sequence minOccurs="0">
                    <element name="objId" type="id"/>
          </sequence>
          <attributeGroup ref="xlink:simpleLink"/>
</complexType>

</schema>
```

FIGURA 5.8 – Descrição do arquivo “sime4.xsd”.

6 Conclusões e Tendências Futuras

Diminuir custos e tempo, além de aumentar a qualidade de produtos e serviços é, hoje, um dos fatores mais importantes em todas as atividades da sociedade que está cada vez mais competitiva. Para os usuários de SIG, a troca de informações entre instituições vem ao encontro desses interesses. Para que seja possível este intercâmbio, os usuários de uma aplicação de SIG precisam ter documentação apropriada que informe a qualidade dos dados que estão recebendo e enviando e, também, quais os tipos de dado que já fazem parte de sua base de dados e quais aqueles que devam gerar uma evolução de esquema.

O sistema de engenharia reversa para SIG, proposto neste trabalho, pode proporcionar ganhos para os usuários de SIG, auxiliando na criação e na manutenção dos modelos conceituais de suas aplicações. O modelo de dados obtido pelo sistema pode servir como base para um modelo mais avançado, definido pelo usuário, e que reflita aspectos da base que não estavam acessíveis ao sistema. O modelo pode, também, ser usado como instrumento de ensino para usuários sem experiência no uso de metodologias de desenvolvimento de sistemas de informação, fato comum entre os projetistas de SIG.

Existem muitos formatos de transferência de dados que são utilizados por produtos de SIG. Seria impossível estudar todos eles neste trabalho. Alguns possuem grande poder de expressão, o que facilita a engenharia reversa. Outros são apenas formatos de arquivos de dados utilizados por determinados *software*. Os formatos que têm maior poder de expressão ainda não são muito utilizados pela comunidade de SIG. Isso porque são propostas mais novas, ainda não adotadas de fato.

Neste trabalho foram estudados três formatos: um formato com pouco poder de expressão, porém muito popular, o Shapefile; um formato com grande poder de expressão, mas não muito utilizado, o SAIF/CSN; e, por fim, um formato que proporciona a troca de informações na WEB e que deve vir a ser muito utilizado futuramente, o GML. Para todos os formatos foi possível estabelecer regras de mapeamento entre eles e o *framework* GeoFrame. Pôde-se, também, verificar que a identificação de regras de mapeamento para outros formatos, que venham a ser estudados, é perfeitamente possível.

Esta pesquisa objetivou definir um conjunto de procedimentos de engenharia reversa de bancos de dados geográficos que auxiliasse nas criação e manutenção de modelos conceituais para aplicações de SIG. Esse objetivo foi alcançado através da arquitetura do sistema proposto e das regras de mapeamento definidas para os diferentes formatos de transferência de dados estudados.

Através do estudo de caso foi possível verificar a eficácia das regras de mapeamento definidas e da arquitetura do sistema proposto. Os dados utilizados para teste foram reais (Projeto SIME) e o modelo que foi obtido é muito semelhante ao verdadeiro. Pode-se constatar, através do estudo de caso, que quanto menor for o poder de expressão do formato de arquivo que esteja sendo processado, maior será a importância de um catálogo de padrões bem definido e, possivelmente, maior também será a intervenção do usuário no processo.

Para que o sistema de engenharia reversa para SIG possa ser completamente detalhado, alguns trabalhos futuros são necessários e imprescindíveis. Um deles é,

certamente, o desenvolvimento de um catálogo de padrões que facilite sua manutenção, estabeleça regras para a escolha dos padrões e garanta a qualidade dos padrões disponíveis. Outro trabalho importante é a construção de um mecanismo que possibilite a criação de uma lista de termos padrão para associar os diferentes nomes dados às entidades do mundo real. Este mecanismo deve ser capaz de identificar sinônimos para nomes de classes e atributos, homônimos com semântica diferente, tanto para interpretações diferentes, como para idiomas diversos.

Por fim, devem ser estabelecidas regras de mapeamento para o maior número possível de formatos de transferência de dados, tornando o sistema o mais abrangente possível.

Anexo 1 Padrões de Modelagem

Neste anexo são apresentados Padrões de Modelagem utilizados nos testes com o protótipo construído. Os padrões estão descritos conforme a estrutura *Problema-Contexto-Forças-Solução* proposta em [LIS2000]. Os diagramas são elaborados com base no *framework* GeoFrame.

- ✓ *Problema – fornece uma declaração sucinta do problema que necessita ser resolvido. Normalmente colocado na forma de pergunta;*
- ✓ *Contexto* – descreve o contexto no qual o problema foi identificado e para o qual a solução foi proposta;
- ✓ *Forças* – conjunto de restrições que foram consideradas quando da elaboração da solução do problema.
- ✓ *Solução* – fornece um diagrama de classes com a solução proposta.

Os padrões 5 e 6 foram extraídos de [LIS2000], enquanto os demais foram extraídos do modelo de dados do SIME.

1. Padrão: Unidades de Referência Espacial

Problema: Quais os elementos presentes na modelagem de unidades de referência espacial de uma região e como se relacionam?

Contexto: As características físicas variam muito nas diversas regiões do Brasil, porém, a estrutura política é organizada da mesma forma. Embora não seja possível reutilizar dados georreferenciados de outras regiões é muito provável que o projeto do banco de dados possa ser reutilizado em grande parte.

Forças:

- ✓ Um estado é constituído de uma ou mais regiões;
- ✓ Uma região é constituída de um ou mais municípios;
- ✓ Um município é constituído de um ou mais distritos e setores censitários;
- ✓ Um distrito é constituído de uma ou mais bairros.

Solução: A Figura 1 mostra o diagrama de classes que compõem o padrão.

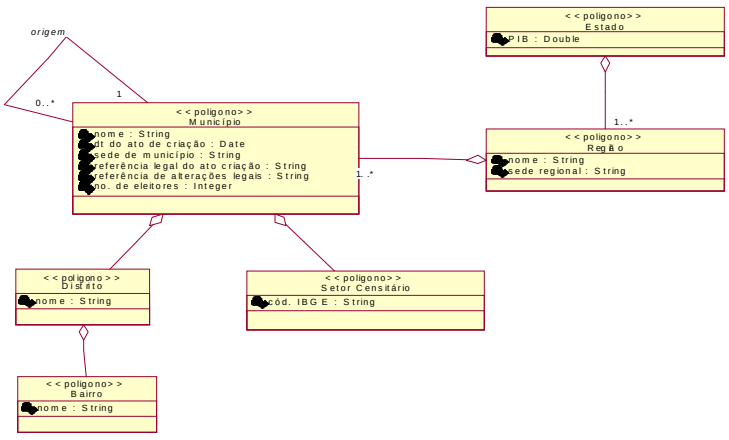

FIGURA 1 – Diagrama de Classes do padrão "Unidades de Referência Espacial".

2. Padrão: Sistema Viário

Problema: Como modelar os elementos de uma sistema viário urbano?

Contexto: No Brasil, praticamente todas as cidades apresentam um mesmo padrão de organização, no qual são estruturadas com base em suas vias de locomoção e os adornos que a elas pertencem.

Forças:

- ✓ Uma via é composta de vários trechos;
- ✓ Os trechos podem possuir um dos seguintes adornos: elemento de sinalização, mobiliário urbano, obra de arte;
- ✓ Um trecho de via está associado a dois nós de trecho de via.

Solução: A Figura 2 mostra o diagrama de classes do padrão.

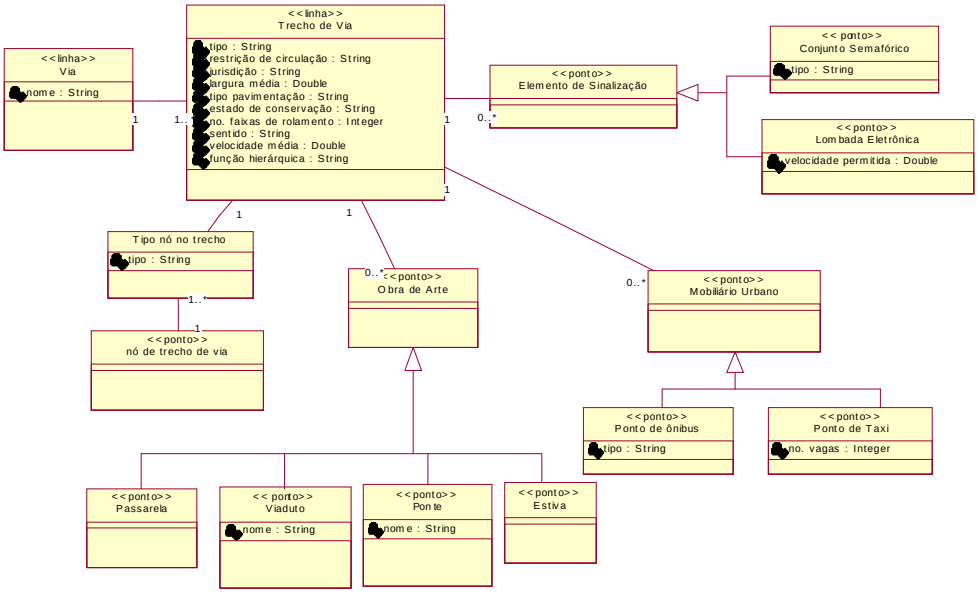

FIGURA 2 – Diagrama de Classes do padrão "Sistema Viário".

3. Padrão: Hidrografia

Problema: Quais os principais elementos presentes na modelagem do tema Hidrografia e como eles estão relacionados?

Contexto: Hidrografia é um tema básico, presente na maioria das aplicações da área de controle ambiental.

Forças:

- ✓ Diferentes terminologias são usadas por usuários de diferentes áreas, inclusive com diferenças regionais (ex.: arroio, córrego, lago, lagoa, riacho);

Solução: A Figura 3 mostra o diagrama de classes que compõem o padrão.

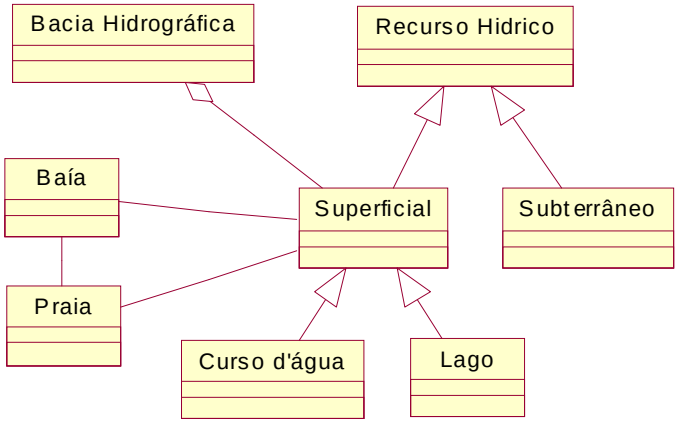

FIGURA 3 – Diagrama de Classes do padrão "Hidrografia".

4. Padrão: Energia

Problema: Como modelar a estrutura de distribuição de energia?

Contexto: A estrutura de distribuição de energia é semelhante independente da companhia que forneça o serviço.

Forças:

- ✓ A linha de transmissão sempre deve estar associada a uma usina geradoura e a uma subestação;
- ✓ A rede de iluminação pública está associada a uma linha de distribuição.

Solução: A Figura 4 mostra o diagrama de classes que compõem o padrão.

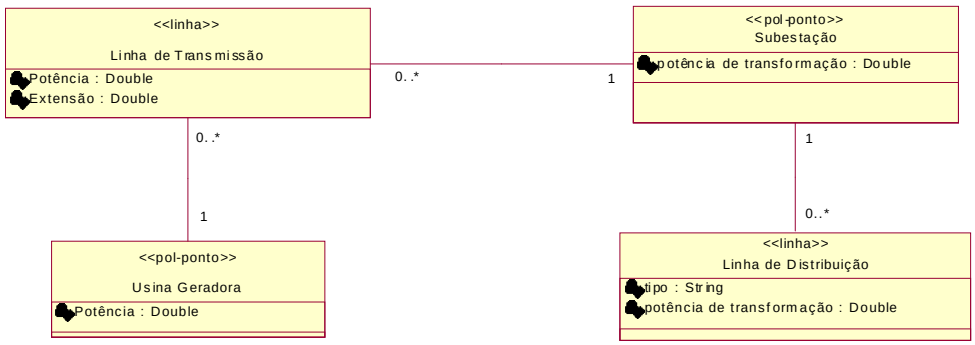

FIGURA 4 – Diagrama de Classes do padrão "Energia".

5. Padrão: Hidrografia

Problema: Quais os principais elementos presentes na modelagem do tema Hidrografia e como eles estão relacionados?

Contexto: Hidrografia é um tema básico, presente na maioria das aplicações da área de controle ambiental. No entanto, é possível obter-se diferentes visões dos elementos pertencentes ao tema como, por exemplo, hidrografia sob o engoque dos diversos tipos de uso de água (ex.: balneários), hidrografia sob o enfoque de meio de transporte fluvial (ex.: rede hidrográfica), hidrografia como conjunto de recursos energéticos (ex.: usinas hidroelétricas e reservatórios).

Forças:

- ✓ Diferentes terminologias são usadas por usuários de diferentes áreas, inclusive com diferenças regionais (ex.: arroio, córrego, lago, lagoa, riacho);
- ✓ Embora não seja um recurso hídrico, o objeto ilha sempre está presente nos diagramas.

Solução: A Figura 5 apresenta um diagrama de classes, descrevendo os principais fenômenos geográficos relacionados com o tema. O projetista deve selecionar apenas as classes de seu interesse e acrescentar novas classes não incluídas no padrão. Também deve adequar a terminologia de acordo com a cultura local.

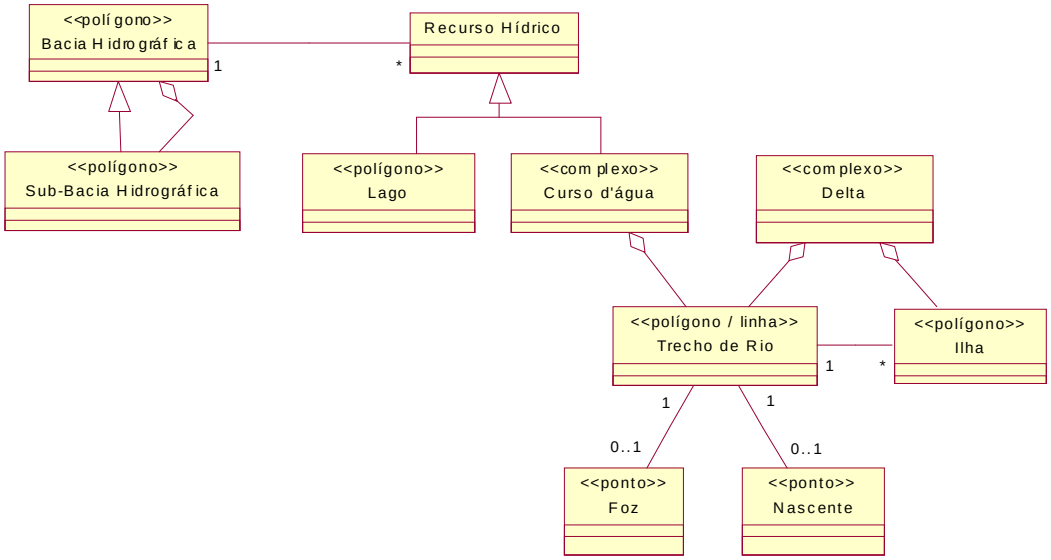

FIGURA 5 – Diagrama de Classes do padrão "Hidrografia" [LIS2000] .

6. Padrão: Malha Viária Urbana

Problema: Como modelar uma malha viária urbana?

Contexto: No Brasil, praticamente todas as cidades apresentam um mesmo padrão de organização, na qual são estruturadas com base em suas vias de locomoção (ex.: ruas, avenidas, travessas). O conjunto de trechos de vias e seus cruzamentos formam uma rede viária urbana.

Forças:

- ✓ Cada via de locomoção, denominada logradouro, deve possuir um código de identificação e um nome, além de estar dividida em diversos trechos;
- ✓ Um trecho de logradouro correspondente ao segmento de via compreendido entre duas conexões, em seqüência, deste com outros logradouros que o cruzam ou interceptam;
- ✓ O conjunto formado pelas conexões (ou pontos terminais) e pelos trechos de logradouros constituem a malha viária urbana.

Solução: A Figura 6 mostra o diagrama de classes que compõe o padrão. Para cada fenômeno geográfico, o padrão especifica apenas os atributos e operações mais genéricos, os quais devem ser estendidos e especializados para cada aplicação

específica. Consequentemente, são especificadas as possíveis abstrações de seus componentes espaciais. Por exemplo, o componente espacial da classe TrechoVia é especificado como sendo linear ou poligonal, embora para um uso específico a mesma classe pode ser especificada com representações distintas.

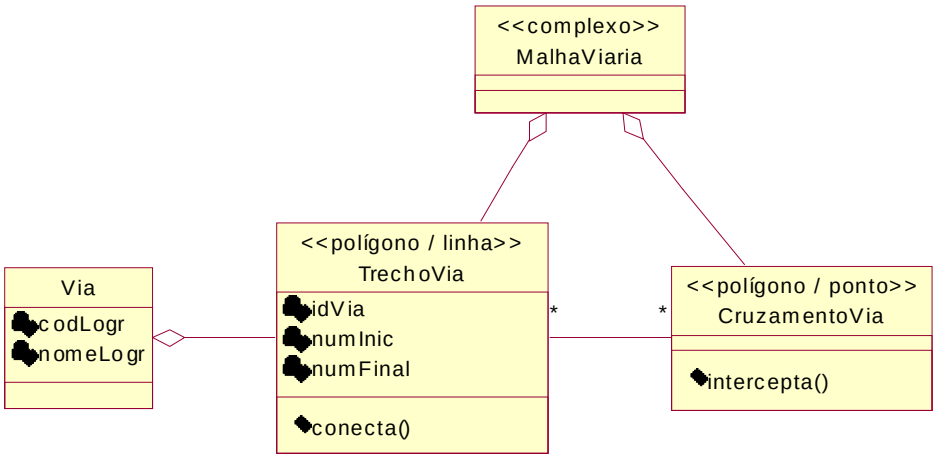

FIGURA 6 – Diagrama de Classes do padrão “Malha Viária Urbana” [LIS2000].

Anexo 2 XML Schema para tema Hidrografia

Neste anexo está um exemplo de XML Schema para definição do modelo de dados baseado no GeoFrame do tema Hidrografia, ilustrado na Figura 2.5. O esquema está escrito de acordo com as normas da GML 2.0 e das regras de mapeamento de esquemas de dados definidos em GeoFrame para GML.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: hidrografia.xsd  -->
<schema targetNamespace="hidrografia" mlns:xlink="http://www.w3.org/1999/xlink"
xmlns:gml=http://www.opengis.net/gml xmlns="http://www.w3.org/2000/10/XMLSchema"
xmlns:hid="hidrografia" elementFormDefault="qualified" version="2.03">
<annotation>
        <appinfo>hidrografia.xsd v2.03 2001-02</appinfo>
        <documentation xml:lang="en">
            GML schema for the hidrography example
        </documentation>
</annotation>
<!-- import constructs from the GML Feature and Geometry schemas -->
<import namespace="http://www.opengis.net/namespaces/gml/core/"
schemaLocation="http://www.opengis.net/namespaces/gml/core/feature.xsd"/>

<!-- ============================================================
      global element declarations
 ============================================================ -->
<element name="tema" type="hid:temaType"    substitutionGroup="gml:_FeatureCollection"/>
<element name="temaMember" type="hid:temaMemberType"
    substitutionGroup="gml:featureMember"/>
<element name="_temaFeature" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="gml:_Feature"/>
<element name="_baciaHidro" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="hid:_temaFeature"/>
<element name="baciaHidro" type="hid:baciaHidroType" substitutionGroup="hid:_baciaHidro"/>
<element name="_recursoHidrico" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="hid:_temaFeature"/>
<element name="recursoHidrico" type="hid:recursoHidricoType"
    substitutionGroup="hid:_recursoHidrico"/>
<element name="_cursoAgua" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="hid:_temaFeature"/>
<element name="cursoAgua" type="hid:cursoAguaType" substitutionGroup="hid:_cursoAgua"/>
<element name="_lago" type="gml:AbstractFeatureType" abstract="true"
    substitutionGroup="hid:_temaFeature"/>
<element name="lago" type="hid:lagoType" substitutionGroup="hid:_lago"/>
<element name="bacia_recurso" type="hid:bacia_recursoType"
    substitutionGroup="hid:_temaFeature"/>
<element name="IdProperty" type="hid:idPropertyType" substitutionGroup="gml:_Feature"/>
<element name="baciaHidroId" type="hid:idPropertyType" substitutionGroup="gml:_Feature"/>
<element name="recursoHidricoId" type="hid:idPropertyType"
    substitutionGroup="gml:_Feature"/>
<element name="cursoAguaId" type="hid:idPropertyType"
    substitutionGroup="hid:recursoHidricoId"/>
<element name="lagoId" type="hid:idPropertyType" substitutionGroup="hid:recursoHidricoId"/>

<!-- ============================================================
      type definitions for region model
 ============================================================ -->
<complexType name="temaType">
        <complexContent>
                <extension base="gml:AbstractFeatureCollectionType">
                        <sequence>
                                <element name="name" type="string"/>
                                <element name="dateCreated" type="date"/>
                        </sequence>
```

```
                </extension>
        </complexContent>
</complexType>

<complexType name="temaMemberType">
        <complexContent>
                <restriction base="gml:FeatureAssociationType">
                        <sequence minOccurs="0">
                                <element ref="hid:_temaFeature"/>
                        </sequence>
                        <attributeGroup ref="gml:AssociationAttributeGroup"/>
                </restriction>
        </complexContent>
</complexType>

<complexType name="baciaHidroType">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <sequence>
                                <element ref="gml:extentOf"/>
                        </sequence>
                        <attribute name="nome" type="string" />
                </extension>
        </complexContent>
</complexType>

<complexType name="recursoHidricoType" abstract="true">
        <complexContent>
                <extension base="gml:AbstractFeatureType">
                        <attribute name="nomeRH" type="string" />
                        <attribute name="tipoRH" type="string"/>
                </extension>
        </complexContent>
</complexType>

<complexType name="cursoAguaType">
        <complexContent>
                <extension base="hid:recursoHidricoType">
                        <sequence>
                                <choice>
                                        <element ref="gml:extentOf"/>
                                        <element ref="gml:centerLineOf"/>
                                </choice>
                        </sequence>
                        <attribute name="vazao" type="number" />
                </extension>
        </complexContent>
</complexType>

<complexType name="lagoType">
        <complexContent>
                <extension base="hid:recursoHidricoType">
                        <sequence>
                                <element ref="gml:extentOf"/>
                        </sequence>
                        <attribute name="volume" type="number" />
                </extension>
        </complexContent>
</complexType>

<complexType name="bacia_recursoType">
        <annotation>
                <appinfo>baciaHidroId--baciaHidroType.fid;recursoHidricoIdId--
cursoAguaType.fid,lagotype.fid</appinfo>
        </annotation>
        <complexContent>
```

```
                <restriction base="gml:FeatureAssociationType">
                        <sequence>
                                <element ref="hid:baciaHidroId"/>
                                <element ref="hid:recursoHidricoId" minOccurs="0"
maxOccurs="unbounded"/>
                        </sequence>
                        <attributeGroup ref="gml:AssociationAttributeGroup"/>
                </restriction>
        </complexContent>
</complexType>

<!-- Id Property -->
<complexType name="idPropertyType">
        <sequence minOccurs="0">
                <element name="objId" type="id"/>
        </sequence>
        <attributeGroup ref="xlink:simpleLink"/>
</complexType>

</schema>
```

Referências

[ANS97] AMERICAN NATIONAL STANDARDS INSTITUTE, INC. **Spatial Data Transfer Standard.** 1997. Disponível em: <http://www.fgdc.gov>. Acesso em: out. 2002.

[BAC2000] BACHMANN, Erik. **Xbase File Format Desciption**. Clickety Click Softwares, 2000. Disponível em: <http://www.e-bachmann.dk/docs/xbase.htm>. Acesso em: mar. 2002.

[BAR99] BARBOSA, C. C. F.; FREITAS, U. M.; CAMARA, G.; YAMADA, M. M. **Formato para Intercâmbio de Dados Geográficos**. Divisão de Processamento de Imagens – INPE, 1999. Disponível em: <http://www.snids.gov.bo/spring/mirror/espanhol/exemplos/ascii_inpe.html>. Acesso em: dez. 2001.

[BAT92] BATINI, C.; CERI, S.; NAVATHE, S. **Database Design:** An Entity-Relationship Approach. [S.l.]:Benjamin/Cummings, 1992.

[BOO2000] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML Guia do Usuário**. Rio de Janeiro: Campus, 2000.

[BOR2000] BORGONY, Vania. **Exercício de Representação do Framework Conceitual GeoFrame no Modelo Abstrato Proposto pelo Consórcio OpenGIS**. 2000. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BRI95] BRITISH COLUMBIA SPECIFICATIONS AND GUIDELINES FOR GEOMATICS. **Spatial Archive and Interchange Format:** SAIF formal definition. Release 3.2. British Columbia, Canada: Ministry of Environment, Lands and Parks, 1995. Disponível em: <http://home.gdbc.gov.bc.ca/SAIF/SAIFHome.html>. Acesso em: mar. 2001.

[CAM96] CÂMARA, Gilberto et al. **Anatomia de Sistemas de Informação Geográfica**. Campinas: UNICAMP, Instituto de Computação, 1996. 193p. Trabalho apresentado na 10. Escola de Computação.

[DAV2000] DAVIS, K.H.; AIKEN, P. H. Data Reverse Engineering: A Historical Survey. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 2000. **Proceedings…** [S.l.:s.n.], 2000.

[ELM99] ELMAGARMID, A.; RUSINKIEWICZ, M.; SHETH, A. **Management of Heterogeneous and Autonomous Database Systems**. San Francisco, California: Morgan Kaufmann, 1999.

[ESR98] ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, Inc. **ESRI Shapefile Technical Description**. 1998. Disponível em: <http://gdal.velocet.ca/projects/shapelib/index.html>. Acesso em: ago. 2001.

[FOR2002] FORNARI, M. R. **GML 2.0 – Geography Markup Language**. Porto Alegre: PPGC da UFRGS, 2002. (RP-316).

[FOR2002a] FORNARI, M. R. Mapeamento de modelos conceituais orientadoa a objetos para esquema em Geography Markup Language (GML). In:

CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 2., Itajaí, SC. **Anais...** Itajaí:[s.n.], 2002.

[HAI2002] HAINAUT, A.-L. et al. Database Reverse Engineering – A Case Study. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 2002. **Proceedings…** [S.l.:s.n], 2002.

[HEU98] HEUSER, C. A. **Projeto de Banco de Dados**. Porto Alegre: Sagra Luzzato, 1998.

[LIS97] LISBOA FILHO, Jugurta. **Modelos Conceituais de Dados para Sistemas de Informação Geográfica**. 1997. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[LIS2000] LISBOA FILHO, Jugurta. **Projeto Conceitual de Banco de Dados Geográficos através da Reutilização de Esquemas, utilizando Padrões de Análise e um Framework Conceitual**. 2000. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[MAT2000] MATTÉ, L. C. **Levantamento de Requisitos de um Banco de Dados Geográficos para Cadastro Imobiliário**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[MUL2000] MÜLLER, H. et al. **Reverse Engineering**: A Roadmap. New York: ACM Press, 2000. p.47-60.

[OGC99] OPEN GIS CONSORTION. **Topic 0, the OpenGIS abstract specification overview. Versão 4**. Disponível em: <http://www.opengis.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC2001] OPEN GIS CONSORTION. **Geography Markup Language (GML) v2.0**. Consortion Document Number: 01-029. Feb. 2001. Disponível em: <http://www.opengis.net/gml/01-029/GML2.html>. Acesso em: mar. 2001.

[PRE99] PRETO, A. G. **MetaSIG:** Ambiente de metadados para aplicações de sistema de informações geográficos. 1999. Dissertação (Mestrado em Ciência da Computação) - Instituto Militar de Engenharia, Rio de Janeiro.

[ROC2001] ROCHA, L. V.; IOCHPE, C.; EDELWEISS, N. **O Framework Conceitual GeoFrame. Versão 2.0.** Porto Alegre: PPGC da UFRGS, 2001. (RP-309).

[SAF2001] SAFE SOFTWARE INC. **Feature Manipulation Engine (FME) – User Manual.** 2001. Disponível em: <http://www.safe.com>. Acesso em: mar. 2002.

[XML99] W3C WORLD WIDE WEB CONSORTION. **Namespaces in XML.** W3C Candidate recommendation (14 January 1999). Jan. 1999. Disponível em: <http//:www.w3.org/TR/1999/REC-xml-names-19990114>. Acesso em: ago. 2001.

[XML2000] W3C WORLD WIDE WEB CONSORTION. **XML Schema Part1:** Structures. W3C Candidate recommendation (24 October 2000). Oct. 2000. Disponível em: <http//:www.w3.org/TR/xmlschema-1>. Acesso em: ago. 2001.

Obras Consultadas

[AND94] ANDERSON, M. Extracting an Entity-Relationship Schema from a Relational Database through Reverse Engineering. In: INTERNATIONAL CONFERENCE ON THE ENTITY-RELATIONSHIP APPROACH, ER, 13., 1994. **Proceedings…** [S.l.:s.n.], 1994.

[BOR2001] BORGONY, Vânia. **Incorporando Suporte a Restrições Espaciais de Caráter Topológico ao Modelo Abstrato do Consórcio Open GIS.** 2001. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BOR96] BORGES, K. A. V.; FONSECA, F. T. Modelagem de Dados Geográficos em Discussão. In: GIS BRASIL, 2., 1996, Curitiba. **Anais…** Curitiba:Sagres, 1996. p. 524-533.

[BOR96a] BORGES, K. A. V.; FONSECA, F. T. Sistemas de Informações Geográficas versus Representação Cartográfica: Uma Visão do Futuro. In: CONGRESSO BRASILEIRO DE CARTOGRAFIA, 17., 1995. **Anais...** Salvador: Sociedade Brasileira de Cartografia, 1995. p. 899-902.

[BOO98] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language user guide**. Reading: Addison-Wesley, 1998.

[BOR99] BORGONY, Vania. **Um Estudo sobre o OpenGIS:** A Proposta da OGC para Interoperabilidade e Distribuição em Sistemas de Informação Geográfica. 1999. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[CAM94] CAMARA, G. **Análise de arquiteturas para Banco de Dados Geográficos Orientados a Objetos**. São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais, 1994. Relatório Técnico.

[CAM95] CAMARA, G. **Modelos, linguagens e arquitetura para Banco de Dados Geográficos**. 1995. Tese (Doutorado em Engenharia de Sistemas) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP.

[COS2001] COSTA, A. C. **Mapeamento de Esquemas Conceituais definidos a partir de um Framework de Banco de Dados Geográficos para esquemas Lógicos baseados no Padrão SAIF**. 2001. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[DAV98] DAVIS JUNIOR, C. A. Modelagem de Dados Geográficos (parte I). **InfoGeo**, [S.l.], v.1, n.2, 1998.

[DAV98a] DAVIS JUNIOR, C. A. Modelagem de Dados Geográficos (parte II). **InfoGeo**, [S.l.], v.1, n.3, p. 44-46, 1998.

[ELM94] ELMASRI, R.; NAVATHE, S. B. **Fundamentals of database systems**. 2nd. ed. Reading: Addison-Welsey, 1994.

[ESR98] ENVIRONMENTAL SYSTEMS RESEARCH INSTITUTE, INC. **The Spatial Data Transfer Standard in ARC/INFO**, 1995. Disponível em: <http://www.ersi.com>. Acesso em: ago. 2001.

[FOR93] FORNARI, M. R. **Evolução de esquemas em baco de dados orientados a objetos utilizando versões**. 1993. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[GAL98] GALANTE, R. M.; SANTOS, C. S.; RUIZ, D. D. A. Um Modelo de Suporte à Evolução de Esquemas em Bancos de Dados Orientados a Objetos com o Emprego de Versões. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 13., 1998. **Anais...** Maringá: [s.n.], 1998. p. 303-317.

[GAL99] GALANTE, R. M.; SANTOS, C. S. Evolução de Esquemas Utilizando Versões em Bancos de Dados Orientados a Objetos. In: CONFERÊNCIA LATINOAMERICANA DE INFORMÁTICA, 1999. **Anais...** Assuncion, Paraguai: [s.n.], 1999.

[GAL2000] GALANTE, R. M.; ROMA, A. B. S.; SANTOS, C. S. Operações Primitivas e Complexas na Evolução de Esquemas em Bancos de Dados Orientados a Objetos. In: TERCERAS JORNADAS IBEROAMERICANAS DE INGENIERÍA DE REQUISITOS Y AMBIENTES SOFTWARE, 2000. **Proceedings...** Cancún, México: [s.n.], 2000.

[GAL2001] GALANTE, R. M. **Evolução de Esquemas em Bancos de Dados Orientados a Objetos com o emprego de versões**. 2001. Exame de Qualificação (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[GAR98] GARAFFA, I. M. **Análise da Adequação de uma hierarquia de classes básicas para modelagem conceitual de SIG através de um Estudo de Caso.** 1998. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[HOL2001] HOLZNER, STEVEN. **Desvendando XML**. Rio de Janeiro: Campus, 2001.

[JOH94] JOHANNESSON, P. A metodo for Transforming Relational Schemas into Conceptual Schemas. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 10., 1994. **Proceedings…** Houston, Texas: IEEE Computer Society, 1994. p.190-201.

[LAR2000] LARMAN, Craig. **Utilizando UML e padrões**: uma introdução à análise e ao projeto orientado a objetos. Porto Alegre: Bookman, 2000.

[LIS99] LISBOA FILHO, J.; IOCHPE, C. Specifying Analysis Patterns for Geographic Databases on the Basis of a Conceptual Framework. In: ACM SYMPOSIUM ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 7., 1999. **Proceedings...** Kansas City: ACM Press, 1999. p. 7-13.

[OMG2001] OMG. **Unified Modeling Language Specification**. Versão 1.4. Sept. 2001. Disponível em: <http//:www.omg.org>. Acesso em: out. 2001.

[PET94] PETIT, J. M. Using Queries to Improve Database Reverse Engineering. In: INTERNATIONAL CONFERENCE ON THE ENTITY-RELATIONSHIP APPROACH, ER, 13., 1994. **Proceedings…** [S.l.:s.n.], 1994.

[RAM96] RAMANATHAN, S.; HODGES, J. **Reverse Engineering Relational Schemas to Object-Orientes Schemas**. Mississippi: Department of Computer Sciense da Mississippi State Univerity, 1996. (Technical Report MSU-960701).

[ROM2000] ROMA, A.B.S. **Um modelo para Evolução de Esquemas de Banco de Dados Orientados a Objetos Usando Versões e Operações Complexas**. 2000. Dissertação (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[SIL95] SILBERCHATZ, A.; KORTH, H. F. **Sistemas de bancos de dados**. 2. ed. São Paulo: Makron Books, 1995.

[THO98] THOMÉ, R. **Interoperabilidade em Geoprocessamento:** Conversão entre Modelos Conceituais de Sistemas de Informação Geográfica e comparação com o Padrão OpenGIS. São José dos Campos: INPE, 1998.

[WEB99] WEBER, E.; ANZOLCH, R.;LISBOA FILHO, J.; COSTA, A. C.; IOCHPE, C. **Qualidade de Dados Geoespaciais**. Porto Alegre: PPGC da UFRGS, 1999. (RP-293)