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

Regras para Transformação de Esquemas Conceituais definidos a partir de um Framework de Banco de Dados Geográficos para Esquemas Lógicos de SIG, com base no Padrão SAIF

por
Andreia Castro Costa

Dissertação submetida à avaliação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Cirano Iochpe
Orientador

Prof ª. Dra. Maria da Graça Abrantes
Co-orientadora

Porto Alegre, abril de 2001

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Costa, Andreia Castro

Regras para Transformação de Esquemas Conceituais definidos a partir de um Framework definidos a partir de Banco de Dados Geográficos para Esquemas Lógicos de SIG, com base no Padrão SAIF / por Andreia Castro Costa. - Porto Alegre: PPGC da UFRGS, 2001.

105 f.:il

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001, Orientador: Iochpe, Cirano.

1. Sistema de informação geográfica. 2. Projeto de banco de dados geográfico. 3. Regras de mapeamento. I. Iochpe, Cirano. II. Título.

Agradecimentos

Agradeço a Deus pela oportunidade desta vida, pelo corpo perfeito e saudável e pela capacidade de desenvolvimento pessoal e intelectual que me auxiliam, de alguma forma, para a contribuição do desenvolvimento da nossa humanidade.

Agradeço ao meu esposo Eduardo pela força, compreensão e incentivo.

Aos meus pais, Joel e Ligia, pelo incentivo e compreensão.

Aos amigos, Vânia, Silvia e Jugurta, pelo auxílio e paciência incessantes.

Ao meu orientador, Dr. Cirano Iochpe, pela confiança e por todo o conhecimento transferido.

A minha co-orientadora, Dra. Graça Abrantes, pela colaboração neste trabalho.

À Dra. Zaida Chora, que deu-me a oportunidade de trabalhar ao seu lado em Lisboa, sendo essa uma contribuição decisiva para a realização deste trabalho.

Aos colegas do Instituto da Vinha e do Vinho em Lisboa, pela oportunidade de transformar muits conceitos em realidade prática.

Sumário

Lista de Figuras

TÍTULO: Regras para Transformação de Esquemas Conceituais definidos a partir de um *Framework* de Banco de Dados Geográficos para Esquemas Lógicos de SIG, com base no Padrão SAIF.

Resumo

O objetivo deste trabalho é investigar o projeto lógico de banco de dados aplicado a Sistemas de Informações Geográficas (SIG), mais especificamente, do mapeamento de esquemas conceituais, orientado a objetos, para esquemas lógicos implementados por softwares de SIG comerciais.

A transformação dos esquemas conceituais para os lógicos é realizado através da identificação de um conjunto de regras genéricas de mapeamento de qualquer esquema conceitual de banco de dados geográficos, baseados em um *framework* conceitual, para os esquemas lógicos proprietários dos diversos SIG existentes.

A concretização desta tarefa de transformação é possível mediante a proposta de um ambiente de suporte. Esse ambiente fornece uma estrutura específica, constituída por uma linguagem formal, definida pelo padrão SAIF (*Saptial Archive and Interchange Format*), pela ferramenta FME (*Feature Manipulation Engine*) e pela ferramenta CASE Rational Rose v.2000e.

O conjunto de regras genéricas, elaborado neste trabalho, é composto por dois subconjuntos. O primeiro define regras de correspondência, determinando uma relação entre os conceitos da realidade percebidos pelo *framework* conceitual e a linguagem formal apresentada pelo padrão SAIF. O segundo subconjunto define regras de transformação, onde busca-se mapear os conceitos do paradigma da orientação a objetos para os conceitos relacionais utilizados pela maioria dos softwares de SIG, disponíveis atualmente no mercado.

Com a finalidade de validar a aplicabilidadde deste conjunto de regras de mapeamento e do ambiente de suporte proposto, este trabalho inclui também a implementação de um protótipo, o qual executa a automatização da transformação dos esquemas conceituais para os esquemas lógicos de banco de dados geográficos.

Palavras-chaves: Sistema de Informação Geográfica; Projeto de Banco de Dados Geográficos; Regras de Mapeamento.

TITLE: “Rules for the Transformation from a Conceptual Schemas which was Defined from a Framework of Geographic Database to Logic Schemas of GIS, based on SAIF.”

Abstract

The main goal of this work is to investigate the logical design of database applied to Geographic Information Systems (GIS). The mapping from conceptual schemas, object-oriented, to logical schemas implemented by commercial GIS software is specifically investigated.

The transformation from conceptual to logical schemas is achieved through the identification of a set of generic rules to map any conceptual schema of a geographic database, which is elaborated from a conceptual framework, to the logical schemas which are proprietary of the several GIS available.

In order to enable the transformation task, an environment is proposed in this work. It provides a specific structure composed by a formal language, which is defined by the Spatial Archive and Interchange Format (SAIF) standard, the Feature Manipulation Engine (FME) tool and the Rational Rose CASE tool 2000e version.

The set of generic rules, which is elaborated in this work, is composed by two subsets. The first one defines correspondence rules, which establish a relationship between the reality concepts that are perceived by the conceptual framework and a formal language presented by the SAIF standard. The second subset defines transformation rules, where the main goal is to perform the mapping of the object-oriented paradigm concepts to the relational concepts which are used by most of the GIS software available today.

In order to validate the applicability of the mapping rules and the proposed environment, the implementation of a prototype which performs the automation process of the transformation from the conceptual schemas to logical schemas of geographic database is also included in this work.

Keywords: Geographic Information System; Geographic Database Design; Mapping Rules.

1 Introdução

Em Sistemas de Informação Geográfica (SIG), assim como, em sistemas convencionais, o projeto de um banco' de dados pode ser dividido em três etapas: conceitual, lógico e físico. No nível conceitual, são representados os elementos da realidade que são representados no banco de dados. No nível físico, são definidos os detalhes da implementação dos dados. Como etapa intermediária desse processo, apresenta-se o projeto lógico, que é o enfoque deste trabalho e que representa a tradução dos conceitos de objetos da realidade ao modelo de dados implementado pelo software de SIG.

O limite entre as atividades do projeto lógico e as atividades das outras fases pode variar de acordo com a metodologia adotada. Por exemplo, alguns autores incluem no projeto lógico atividades tais como particionamento e sobreposição de entidades e análises de performance com base no volume de dados associado às aplicações [BAT 92]. Essas atividades, eventualmente, aparecem como parte do projeto físico em outras metodologias. Neste trabalho, considera-se o projeto lógico como uma fase intermediária, que se destina à tradução dos construtores utilizados no esquema conceitual em esquemas lógicos que podem ser, efetivamente, implementados em Sistema de Gerência de Banco de Dados (SGBD). Esquemas conceituais podem ser descritos com base no paradigma de orientação a objetos, que pode incluir os modelos de dados semânticos.

Os conceitos da orientação a objetos podem ser utilizados como mecanismos de abstração da realidade para a geração de esquemas conceituais. Algumas experiências [CLE 94], [PAR 98], [ABR 98], têm mostrado que o conceito de orientação a objetos pode aumentar a integração entre os dados e facilita a interpretação do usuário que participa desta fase da aplicação.

Neste cenário, desenvolveu-se no PPGC da UFRGS, no âmbito do projeto SIGMODA (Análise e Comparação de Metodologias de Modelagem de Dados em Sistemas de Informações Geográficas), um *framework* conceitual denominado GeoFrame [LIS 99], cujo objetivo é auxiliar o projetista na modelagem conceitual de banco de dados geográficos. Porém, para aprimorar a ferramenta de modelagem de banco de dados geográficos, é desejável incluir mais algumas facilidades. Dentre elas, destaca-se o mapeamento de esquemas conceituais para esquemas lógicos proprietários dos diversos SIG.

Dessa forma, o objetivo desta pesquisa é elaborar um conjunto de regras de mapeamento de esquemas conceituais, baseados no GeoFrame, para esquemas lógicos de SIG. Para tanto, utiliza-se as abordagens de algumas ferramentas que são selecionadas para auxiliar no desenvolvimento de todo o processo.

Como não existe um modelo lógico de SIG que seja único e, muito menos, uma regra única para o mapeamento de esquemas, busca-se elaborar um conjunto de regras genéricas para realizar o mapeamento de qualquer esquema conceitual para uma linguagem formal, definida por um padrão de intercâmbio e armazenamento de dados espaciais.

A linguagem formal, utilizada nesta pesquisa, permite a conversão de esquemas conceituais para os esquemas lógicos de alguns SIG, através da utilização de uma ferramenta de tradução de dados espaciais. Porém, essa linguagem apresenta algumas limitações que são supridas no decorrer deste trabalho.

Com a finalidade de validar a aplicabilidade das regras de mapeamento, este trabalho inclui também a definição de um ambiente de suporte a este processo. Esse ambiente fornece uma estrutura adequada, constituída por ferramentas já existentes e adaptadas à realização da tarefa de mapeamento de esquemas, através da utilização de regras genéricas. As regras e o ambiente de suporte definidos neste trabalho são demonstrados através do desenvolvimento de um protótipo. Como resultado final, torna-se possível realizar o mapeamento dos esquemas conceituais, elaborados a partir do *framework* GeoFrame, para os esquemas lógicos de alguns softwares de SIG comerciais.

1.1 Motivação

Dados geográficos possuem características peculiares que fazem com que sua modelagem seja mais complexa do que a de dados convencionais. Modelar os aspectos espaciais é de fundamental importância na criação de um banco de dados geográficos, principalmente porque está se tratando com uma abstração da realidade geográfica, onde a visão que os usuários têm do mundo real pode variar, dependendo do que eles necessitam representar e do que eles esperam extrair desta representação.

Assim, pelas peculiaridades que os SIG apresentam, torna-se necessário introduzir adaptações em metodologias e ferramentas de modelagem de dados, a fim de que essas melhor capturem as características dos dados geográficos. Existem várias propostas de abordagem para resolver os problemas de modelos conceituais para SIG. Como exemplos, pode-se citar os seguintes modelos de dados: modelo *Clementini* [CLE 94]; modelo *GeoOOA* [KÖS 95]; modelo *Geo-OMT* [ABR 98]; *framework* GeoFrame [LIS 99].

Normalmente, tais modelos não são adotados pelos projetistas de banco de dados geográficos. Este fato ocorre, principalmente, devido à falta de ferramentas *CASE* (*Computer-Assisted Software Engineering*) que realizem o mapeamento de esquemas de SIG, elaborados com base nesses modelos conceituais, para os modelos de dados implementados pelos produtos comerciais de SIG atuais. Somando-se a este fato, ocorre que, muitas organizações por deterem apenas o conhecimento de um SIG, acabam por utilizar, para efeitos de modelagem, conceitos, primitivas e terminologia específicos daquele sistema, ou seja, projetam o banco de dados diretamente no modelo lógico implementado pelo SIG comercial.

Neste contexto, surgiu a proposta de um primeiro trabalho individual [COS 99], que elaborou regras de mapeamento entre esquemas conceituais, baseados na BCH/GIS (Hierarquia de Classes Básicas para Sistemas de Informações Geográficas) [LIS 98], em um modelo lógico intermediário, baseado no modelo de *Hadzilacos* [HAD 96]. Esse modelo intermediário poderia ser traduzido para os modelos lógicos de alguns softwares de SIG. Porém, durante a avaliação desta proposta, percebeu-se que seria muito complexa a tarefa de elaborar um conjunto de regras específicas para cada SIG, devido as diferenças internas de cada modelo.

Então, através da pesquisa, foi identificada a ferramenta *FME* (*Feature Manipulation Engine*) que trabalha como uma grande central de traduções de diversos arquivos e softwares de SIG. Em um estudo preliminar, verificou-se que esse tradutor utiliza um padrão, denominado *SAIF* (*Spatial Archive and Interchange Format*), como uma das possíveis origens para realizar traduções entre softwares de SIG.

Desta forma, a pesquisa direcionou-se para o detalhamento do *SAIF*, documentado no segundo trabalho individual [COS 2000]. Este trabalho buscou analisar a possibilidade de manter uma correspondência entre alguns conceitos das classes do padrão de intercâmbio de dados espaciais e da evolução da *BCH/GIS* [IOC 98], atualmente denominada *framework* conceitual GeoFrame [LIS 99].

Percebendo a dificuldade em elaborar regras de mapeamento para cada modelo de SIG, conhecendo as potencialidades da ferramenta de tradução de dados e a possibilidade de correspondência entre conceitos da realidade percebidos pelo diagrama de classes básicas do GeoFrame e do padrão de intercâmbio de dados, verificou-se a possibilidade de elaborar um conjunto de regras de mapeamento de esquemas conceituais, baseados no *framework* GeoFrame, para os esquemas lógicos de alguns softwares SIG, através da utilização dessas ferramentas disponíveis no mercado.

Assim, a principal motivação desta pesquisa é o *framework* GeoFrame, pois além de apresentar um diagrama de classes que facilita a modelagem conceitual de dados geográficos, os esquemas produzidos, usando esse modelo, tendem a ser mais compactos que os demais, devido ao conteúdo semântico de suas primitivas de modelagem.

No entanto, a escolha pelo GeoFrame se justifica essencialmente porque, associa a modelagem conceitual à representação de classes e propõe uma clara integração entre representação de campo e objetos, tornando um esquema conciso.

Esse *framework* desencadeou todo o processo de pesquisa deste trabalho, onde são elaboradas regras de mapeamento para os esquemas conceituais desenvolvidos a partir do GeoFrame para os modelos lógicos que os softwares de SIG comerciais implementam.

1.2 Objetivos

O objetivo principal desta pesquisa é a elaboração de um conjunto de regras genéricas de mapeamento de esquemas conceituais, elaborados a partir do GeoFrame, para esquemas lógicos que os softwares de SIG comerciais implementam, abrindo caminho para a construção de ferramentas *CASE* para projeto de banco de dados geográficos.

Em função da complexidade desse objetivo, investiga-se a possibilidade da utilização de algumas ferramentas já existentes, as quais são utilizadas neste trabalho como intermediárias no processo de mapeamento entre esquemas.

O desenvolvimento metodológico deste trabalho segue uma estrutura definida para alcançar o objetivo proposto, baseando-se principalmente na existência de uma ferramenta de tradução de dados espaciais - *FME*.

Essa ferramenta é capaz de executar traduções entre esquemas, partindo da utilização de alguns softwares de SIG ou de uma linguagem formal estruturada, a qual é parte integrante de um padrão de intercâmbio e armazenamento de dados espaciais.

Dada a opção da estrutura acima, podem-se destacar como objetivos específicos: a compreensão do *framework* conceitual; a descrição de uma ferramenta de tradução entre modelos de dados espaciais; a especificação de um padrão de intercâmbio e armazenamento de dados espaciais e de sua linguagem formal que, utilizados de forma conjunta, possibilitam a verificação da funcionalidade das regras de mapeamento elaboradas.

Uma das características fundamentais desta pesquisa é a busca de uma solução que pode ser utilizada, na prática, pelo projetista de SIG, para a geração de um esquema lógico de dados geográficos, a partir do esquema conceitual da aplicação.

1.3 Trabalho Relacionado

Esta dissertação insere-se na área de *banco de dados não convencionais*, um tópico considerado recente em *sistemas de informação*. Por tratar-se de uma área em consolidação, o contexto é dinâmico, logo, sujeito a discussões e questionamentos entre as diversas visões. Esta Seção apresenta uma descrição de um trabalho considerado relevante para o propósito desta dissertação, embora com enfoque diferenciado do apresentado aqui.

Durante a pesquisa por trabalhos na literatura e na rede de computadores, tomou-se conhecimento do trabalho realizado por Bédard [BED 99]. Nele, o autor apresenta uma ferramenta *CASE* denominada *Perceptory*, a qual realiza tradução de esquemas conceituais para o esquema lógico de um software de SIG, através de regras de mapeamento, tornando-se uma proposta bastante semelhante ao objetivo deste trabalho. Porém, o *Perceptory* implementa seu próprio modelo conceitual e requer a utilização da ferramenta *visio2000*, uma aplicação gráfica para suporte à criação de diversos tipos de diagramas.

O *Perceptory* está em desenvolvimento na Universidade Laval-Canadá e tem como objetivo modelar dados espaciais e espaço-temporais [BED 99]. A base dessa ferramenta é um conjunto de conceitos e construtores, denominado *PLVs* (*Plug-in for Visual Language*), que é o resultado da evolução do trabalho de um grupo de pesquisa na área de modelagem e desenvolvimento de bases de dados espaciais.

Essa ferramenta baseia-se em conceitos que se encontram na área de orientação a objetos e utiliza a sintaxe da linguagem UML [BOO 98]. Especificamente, o *Perceptory* permite ao usuário criar modelos de classes espaciais e espaço-temporais, documentar esses modelos através do dicionário de dados, produzir relatórios com base no dicionário, criar arquivos .htm para consultas na *web* e gerar código para o *oracle8i* através do módulo espacial.

A geração de código exige a utilização de regras de mapeamento para um SIG específico (*oracle8i*) [BED 2000]. Portanto, alguns conceitos são abstraídos desse

processo, a fim de serem incluídos na elaboração de regras de mapeamento descritas no Capítulo 3.

Segundo Bédard [BED 99], essa ferramenta *CASE* vem buscando cumprir às normas da *ISO/TC211* [ISO 98], bem como respeitar os resultados produzidos pelo *OpenGIS Consortium*. Um dos avanços considerados importantes é relativo ao repositório de dados geoespaciais, os quais são utilizados para armazenar informações conceituais das aplicações e de seus respectivos dicionários [BRO 2000].

1.4 Contribuições

As principais contribuições deste trabalho são:

- a elaboração de um conjunto de regras genéricas de mapeamento de esquemas conceituais geográficos, a partir do *framework* conceitual GeoFrame, para os esquemas lógicos de alguns softwares de SIG;
- a construção de uma estrutura de software para fornecer suporte às regras de mapeamento, através de ferramentas existentes e adaptadas às necessidades dos objetivos propostos;
- o desenvolvimento de um protótipo que permite validar a aplicabilidade e a eficácia das regras de mapeamento, bem como do ambiente de suporte proposto;
- a proposta de uma ferramenta que pode ser utilizada pelo projetista de sistemas de informação geográfica na elaboração do esquema lógico de suas aplicações a partir de um esquema conceitual baseado no GeoFrame.

As contribuições obtidas nesta dissertação estão demonstradas através de um exemplo, que procura modelar parte de um projeto de aplicação geográfica denominado SIGPROGB[1], desde o esquema conceitual até a nível lógico, apoiado em um SIG comercial.

Essas contribuições diferenciam-se do trabalho descrito na Seção 1.3 por propor um enfoque baseado em regras genéricas de mapeamento para alguns softwares de SIG, através da utilização de um padrão de intercâmbio de dados espaciais. Em consequência, o conjunto de regras proposto fortalece a utilização do *framework* conceitual GeoFrame, reduzindo a distância existente entre os projetistas de SIG e o modelo, e ainda colabora significativamente para a realização de traduções entre os esquemas conceitual e lógico de banco de dados geográficos.

[1] SIGBPROGB envolve a modelagem conceitual do SIG do Programa Pró-Guaíba e tem como objetivo o desenvolvimento de uma série de ações visando a melhoria da qualidade da água em toda a bacia hidrográfica do Rio Guaíba.

1.5 Estrutura da Dissertação

O restante desta dissertação está organizado como segue.

O Capítulo 2 é divido em três Seções. A primeira seção apresenta uma revisão de conceitos considerados importantes para definir o contexto onde está inserido o tema da dissertação. A segunda seção descreve os requisitos necessários para realizar o mapeamento, sendo descrito o *framework* conceitual GeoFrame e o padrão de intercâmbio e armazenamento de dados espaciais denominado *SAIF*. A última seção descreve as ferramentas necessárias para a elaboração de um ambiente de suporte que possibilite a validação das regras de mapeamento.

No Capítulo 3 são especificados dois conjuntos regras utilizados para concretizar o mapeamento entre o *framework* conceitual GeoFrame e o padrão de intercâmbio e armazenamento de dados espaciais.

O Capítulo 4 descreve o protótipo de um ambiente de suporte para aplicação das regras de mapeamento e exemplifica-o através da utilização de parte do projeto SIGPROGB.

Finalmente, no Capítulo 5, são apresentadas as conclusões finais e as dificuldades encontradas para a elaboração deste trabalho.

2 Visão Geral de Banco de Dados Geográficos

O objetivo deste Capítulo é introduzir alguns conceitos considerados relevantes para o propósito do trabalho, constituindo uma base preliminar de estudos para situar o tema em um contexto mais específico.

A Seção 2.1 tem por finalidade localizar o leitor no núcleo de todo o processo de pesquisa, sendo descrito o projeto de banco de dados geográficos; uma visão geral do paradigma orientado a objetos e da linguagem UML. Em seguida, é apresentada a descrição sucinta da ferramenta *Perceptory*, a qual representa um trabalho semelhante ao propósito desta pesquisa.

Na Seção 2.2 são descritos os requisitos básicos para elaboração das regras de mapeamento definidas no Capítulo 3.

Na Seção 2.3 são apresentadas duas ferramentas de suporte, as quais são utilizadas na implementação do protótipo definido no Capítulo 4.

2.1 Conceitos Básicos

Um primeiro conceito importante é o de processo de desenvolvimento de um BDG (Banco de Dados Geográficos). Esse processo está intimamente ligado ao desenvolvimento de um banco de dados convencional, onde as informações que comporão o BDG são especificadas, utilizando-se diferentes níveis de abstração, desde o nível conceitual até o nível (físico) de armazenamento dos dados [PEU 93]. Esses níveis são descritos detalhadamente na Seção 2.1.1**.** A Seção 2.1.2 caracteriza brevemente a modelagem orientada a objetos em SIG e a linguagem UML (*Unified Modeling Language*) [BOO 98]. Na Seção 2.1.3, é apresentada uma visão geral da ferramenta *Perceptory* [BED 99] juntamente com um dos processos considerados relevantes para este trabalho.

2.1.1 Fases do Projeto de Banco de Dados Geográficos

O projeto de banco de dados [BAT 92] é o processo que determina a organização estrutural de um banco de dados, a partir dos requisitos de dados previamente especificados. Esse projeto realiza transformações e pode ser dividido em fases. Segundo Lisboa [LIS 99], o projeto de um BDG pode ser dividido em três fases: (a) projeto conceitual; (b) projeto lógico; (c) projeto físico. Esse processo é interativo, a saída de uma fase é tomada como entrada para a fase subseqüente. A seguir, na Figura 2.1 são ilustradas as fases do projeto de banco de dados, juntamente com as entradas e as saídas de cada fase.

O *Projeto Conceitual* tem como objetivo representar os objetos da realidade que são representados no banco de dados e os relacionamentos existentes entre eles. Nesta fase, a modelagem deve ser acompanhada por detalhes de como os objetos geográficos são representados espacialmente [GAR 98]. O modelo conceitual é descrito em um alto

nível de abstração (ex.: modelo orientado a objetos), e não considera os aspectos de implementação, o que o torna independente do software a ser utilizado.

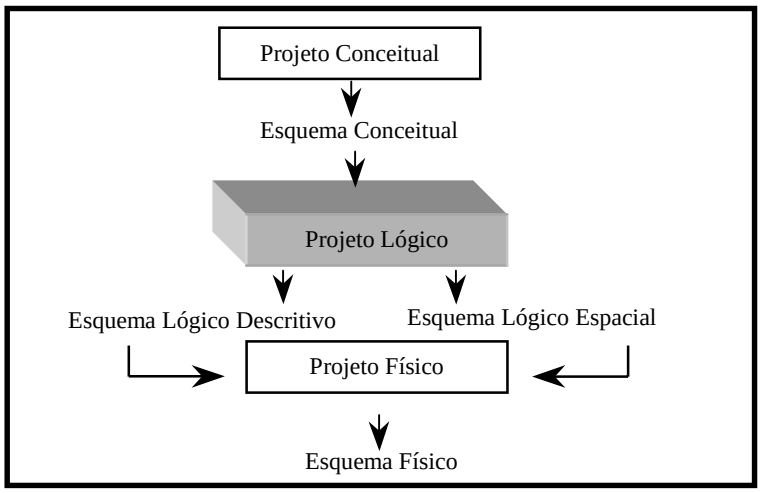


FIGURA 2.1 – Fases do projeto de banco de dados geográfico [LIS 99]

A análise conceitual de dados é uma fase necessária devido à complexidade dos dados geográficos, em termos dos relacionamentos a serem modelados e das operações a serem executadas. Algumas características de um projeto conceitual são: independência do software e hardware; descrição e definição dos objetos da realidade a serem considerados.

As necessidades de cada aplicação geográfica com relação à representação de objetos são determinadas durante o processo de modelagem de dados. Neste processo, adota-se um determinado modelo de dados, um conjunto de conceitos com semântica suficientemente poderosa para formular a estrutura do banco de dados geográficos, incluindo os tipos de dados, os relacionamentos e as restrições que se aplicam aos dados. Ao final deste processo, conforme ilustra a Figura 2.1, tem-se uma descrição coerente desta estrutura, denominada *esquema conceitual*.

A modelagem de dados das aplicações geográficas difere da convencional, devido às características espaciais e à natureza dos relacionamentos entre as entidades geográficas.

O *Projeto Lógico* converte o esquema conceitual para um esquema lógico de dados. Essa transformação varia de acordo com o modelo de dados do software de SIG adotado para a implementação da aplicação. Como a maioria dos SIG disponíveis comercialmente ainda utilizam uma arquitetura dual, ou seja, possuem dois componentes distintos, um para gerenciamento de dados espaciais e outro para os descritivos, a fase do projeto lógico pode ser dividida em duas subfases: *projeto lógico descritivo* (PLD - emprega as técnicas convencionais de BD) e *projeto lógico espacial* (PLE - faz mapeamento dos objetos geográficos para o esquema específico do software de SIG a ser empregado na implementação). Os dados contidos no PLD fazem referência aos dados contidos em PLE e vice-versa.

Para um software com arquitetura integrada, onde as estruturas de dados são armazenadas em um único gerenciador de dados, o projeto lógico tem uma única fase.

Exemplos dessa arquitetura são os softwares *Smallword* e o *MGE/Intergraph*, a partir da versão 7.0 do SGBD *Oracle SDO*.

Embora a maioria dos softwares adote a arquitetura dual, onde os atributos descritivos e espaciais são armazenados separadamente, neste trabalho a tradução desses itens para o esquema lógico é realizada em conjunto e introduzida no software de SIG de forma dual.

No *Projeto Físico* define-se os detalhes de implementação de dados (ex.: meios de armazenamento e índices de acesso) e as estruturas de dados espaciais que são utilizadas. Esse projeto depende tanto do software como do hardware escolhidos e requer considerações sobre as estruturas internas de armazenamento de SGBD e das propriedades dinâmicas da aplicação (tipo e freqüência das consultas e atualizações sobre o BD).

2.1.2 Modelagem Orientada a Objetos em SIG

Na área de banco de dados, sabe-se que os modelos de dados semânticos (por exemplo, Entidade Relacionamento – E-R) não são suficientes para descrever os dados das chamadas "novas aplicações", tais como CAD e SIG. Os dados dessas aplicações são muito complexos e evolutivos, sendo necessária uma certa adequação dos modelos existentes [HAD 96], [PIR 93].

Para compensar essas deficiências, várias abordagens têm sido propostas como, por exemplo, a abordagem orientada a objetos (O.O.). Esses modelos são mais expressivos e flexíveis que o E-R. O modelo E-R foi criado para permitir a representação de vários problemas, usando um pequeno conjunto de conceitos simples, enquanto que os modelos O.O. foram projetados para a criação e representação de estruturas bem mais complexas de uma maneira coerente e uniforme [SMI 97].

O uso de conceitos de O.O. em geoprocessamento foi analisado pela primeira vez em Egenhofer [EGE 87]. Neste trabalho, o autor mostra como aplicar conceitos de classificação, especialização, generalização e agregação ao projeto de SIG. Uma revisão mais recente dos resultados teóricos em uso de O.O. em projeto de SIG, pode ser visto em Egenhofer [EGE 95].

Uma das principais linhas de pesquisa derivada da comunidade de banco de dados procura estender os conceitos O.O. ao domínio geográfico. Merecem destaque os seguintes modelos:

- modelo Clementini – desenvolvido na Universidade de L´Aquila, Itália. Organiza o conhecimento sobre a aplicação geográfica em termos dos conceitos básicos do paradigma de orientação a objetos – classes, instâncias e métodos [CLE 94];

- modelo MADS – adota o paradigma de orientação a objetos, através da inclusão de características do modelo padrão ODMG (*Object Database Management Group*). Esse modelo foi desenvolvido na Universidade de Lausanne, Suíça [PAR 98].

- modelo GeoOOA - estende o modelo usado na metodologia de análise orientada a objeto de Coad/Yourdon [COA91] para análise de requisitos de aplicações

geográficas. O modelo GeoOOA serviu de base para a implementação da ferramenta CASE *REGIS* [ISO 99], que emprega a notação gráfica da linguagem UML (*Unified Modeling Language*) [BOO 98]. O modelo GeoOOA foi desenvolvido na Universidade de Hagen, Alemanha [KÖS95];

- modelo Geo-OMT – estende a estrutura dos conceitos OMT para possibilitar a construção de modelos adequados à representação das características específicas dos sistemas que incluem dados geográficos [ABR 98]. O Geo-OMT foi desenvolvido na Universidade Técnica de Lisboa, Portugal.

Dessa forma, um banco de dados pode ser visto como um modelo abstrato de uma porção da realidade, uma vez que, seus dados representam um subconjunto de elementos pertencentes a essa realidade [BED 99]. Abstrair uma porção da realidade para projetar um banco de dados implica em selecionar as entidades da realidade que são significativas, determinar suas características essenciais para que o seu funcionamento seja adequado ao sistema, identificando como elas podem ser estruturadas juntamente com seus relacionamentos.

O processo de modelagem conceitual é realizado, utilizando-se o mecanismo de abstração. Esse mecanismo é uma capacidade de visão em alto nível, que permite examinar problemas para selecionar grupos comuns, encontrar generalidades para melhor compreender o problema e construir esquemas de banco de dados.

Para auxiliar na construção de modelos através da abstração, característica essencial da O.O., foram desenvolvidos alguns mecanismos que são encontrados em [RUM 91]. A apresentação desses mecanismos está fora do escopo deste trabalho.

Em abordagens O.O., existem diversas linguagens para especificação de modelos de dados. Entre as mais conhecidas pode-se citar: OOA [COA 91], OMT [RUM 91] e UML [BOO 98].

Atualmente, a linguagem mais utilizada é o padrão UML , cuja eficácia é reforçada, no caso de projeto de aplicações de SIG, pelo surgimento de ferramentas *CASE* como, por exemplo, *Perceptory* [BED 99] e *REGIS* [ISO 99].

Uma das vantagens da utilização da UML é a possibilidade de elaborar e empregar uma variedade de construtores da linguagem, especialmente esteriótipos. Esses esteriótipos podem ser estendidos pelo projetista da aplicação e podem ser empregados como qualquer outro elemento da linguagem. Torna-se importante destacar que o esteriótipo é um mecanismo de extensão da UML. Uma descrição detalhada da linguagem UML pode ser encontrada em [BOO 98].

A ferramenta *Perceptory*, descrita na Seção seguinte, exige que o esquema conceitual esteja de acordo com a definição *PLVs* (*Plug-in for Visual Language*), originada do modelo *Modul-R* [BED 96], que é uma adaptação à UML para SIG. A UML foi utilizada também no desenvolvimento do *framework* conceitual GeoFrame [LIS 99], descrito na Seção 2.2.1.

2.1.3 Perceptory – Uma Ferramenta para Suporte À Modelagem Orientada a Objetos em SIG

Como referido anteriormente, esta Seção tem como objetivo descrever um dos processos que compõem a ferramenta. Dessa forma, o enfoque é dado para a geração de código para um software específico de SIG (*oracle8i*). Essa geração implica na elaboração e utilização de regras de mapeamento. Portanto, alguns conceitos são abstraídos desse processo, a fim de serem incluídos na elaboração de regras de mapeamento descritas no Capítulo 3.

O processo de geração de código é executado através de um item do menu denominado *model traductor,* presente na ferramenta. Como resultado, o tradutor fornece ao usuário um *script* em linguagem SQL, formato de entrada do *oracle8i*, que define o esquema da base de dados.

No final da execução do processo, são apresentados, ao usuário, quatro arquivos distintos:

- oracle8i SQL script (*script.sql*);
- restrições de integridade (*script.com*);
- lista das operações executadas pelo gerador de código (*script.log*);
- operações em tabelas de resíduo (*script.del*).

Esses arquivos são gerados devido às exigências impostas pela estrutura do *oracle8i*. Enquanto o mundo do software convencional apresenta soluções em que cada produto permite acessar uma grande quantidade de diferentes formatos de arquivos com as mesmas características, os correspondentes componentes de SIG têm se mantido com soluções proprietárias.

Dessa forma, verifica-se que cada SIG implementa o seu próprio modelo lógico, sendo necessário implementar um gerador de código para cada software. Esse fator, conforme o próprio autor descreve, exige que o *Perceptory* implemente diferentes *model traductor* com diferentes regras de mapeamento, dependendo da estrutura do software de SIG que se deseja mapear. Atualmente, essa ferramenta está buscando implementar regras para os modelos lógicos do *ArcView, ArcInfo, MapInfo* [BED 2000].

As regras de mapeamento do esquema conceitual para o esquema lógico do *oracle8i,* estão embutidas no programa que executa o mapeamento automaticamente, restando compreendê-las através do *model traductor.*

Através da investigação desse módulo, torna-se possível verificar a existência de algumas situações definidas, através de regras, para gerar a linguagem SQL. Abaixo, estão descritas as situações encontradas nesse módulo:

- verificação do *Id* – cada tabela é identificada, através da definição de um *Id* para cada classe definida pelas *PLVs*;

- relação 1:1 – neste ponto o usuário tem a opção entre dois mapeamentos: junção das classes da relação em uma única tabela ou armazenamento da informação em tabelas individuais;

- implementação da generalização – este passo permite a seleção da melhor alternativa de implementação da hierarquia de classes. O gerador de código oferece quatro opções:

    - migrar os subtipos para o supertipo, instanciando somente o supertipo;
    - aplicar herança e instanciar somente os subtipos;
    - aplicar herança e instanciar o supertipo e os subtipos;
    - adicionar os *Id´s* do subtipo no supertipo e instanciar ambos, sem considerar o conceito de herança.

- referências espacial e temporal – essas referências, nos *PLVs*, são representadas por esteriótipos criados em UML [BOO 98];

- geração de chave estrangeira – essa geração visa proporcionar a fusão de tabelas que estão associadas através do relacionamento ´n:N´. Esse tipo de multiplicidade permite a seleção de identificadores;

- elaboração do *script* – nesse passo é possível determinar o número máximo de caracteres para a definição das tabelas e dos atributos das tabelas SQL .

A descrição detalhada do *model traductor* da ferramenta *Perceptory* foge ao escopo deste trabalho, devido à falta da ferramenta *oracle8i* para testes. Essa ferramenta torna-se indispensável, pois sem ela torna-se difícil o entendimento dos comportamentos de cada um dos seus componentes. Essa dificuldade é mais acentuada, principalmente, no item relativo às referências espacial e temporal.

2.2 Requisitos Básicos para Elaboração de Regras

Conforme referido na Seção 1.2 o objetivo desta dissertação é elaborar regras de mapeamento que buscam traduzir esquemas conceituais, baseados no *framework* conceitual GeoFrame, para esquemas lógicos de alguns softwares de SIG. Esse mapeamento é realizado através da utilização de um padrão de armazenamento e intercâmbio de dados espaciais.Para melhor entendimento do trabalho, a Seção 2.2.1 descreve os principais conceitos presentes no *framework*[1] GeoFrame que é o ponto de partida para todo o processo de mapeamento, bem como suas primitivas para representação de componentes espaciais. A Seção 2.2.2 caracteriza o padrão *SAIF* [SAI

[1] *Framework* pode ser definido como um projeto genérico em um domínio que pode ser adaptado a aplicações específicas, servindo de molde para a construção de aplicações [LIS 00a].

95], o qual é utilizado como intermediário no processo de mapeamento entre os esquemas conceitual, baseado no GeoFrame, e os modelos lógicos de ferramentas comerciais de SIG.

2.2.1 Framework Conceitual GeoFrame

GeoFrame é um *framework* conceitual que fornece um diagrama de classes básicas para auxiliar o projetista, tanto na modelagem conceitual de dados geográficos, como na especificação de padrões de análise em banco de dados geográficos [LIS 99].

Com o objetivo de aprimorar a ferramenta de modelagem conceitual de banco de dados geográficos, é necessário acrescentar mais alguns recursos como, por exemplo, elaboração de regras para realizar o mapeamento de esquemas conceituais para esquemas lógicos de alguns softwares de SIG. Dessa forma, é de suma importância apresentar o ponto inicial de todo o processo. Cabe destacar que, o objetivo da pesquisa não é o detalhamento do *framework*, cuja descrição completa encontra-se em [LIS 2000].

Esse *framework* parte das primitivas definidas para o diagrama de classes da UML [BOO 98], introduzindo primitivas geográficas com o objetivo de aumentar a capacidade de representação semântica do modelo, reduzindo assim a distância entre o modelo mental do espaço a ser modelado e o modelo de representação usual. Além disso, o GeoFrame permite a especificação de atributos alfanuméricos e pode ocorrer a definição de métodos associados para cada classe. Suas principais qualidades são: sua expressividade gráfica e sua capacidade de representação, representando a dinâmica da interação entre as diversas entidades espaciais e não espaciais. Na Figura 2.2 é ilustrado o diagrama de classes do GeoFrame e suas classes são descritas a seguir.

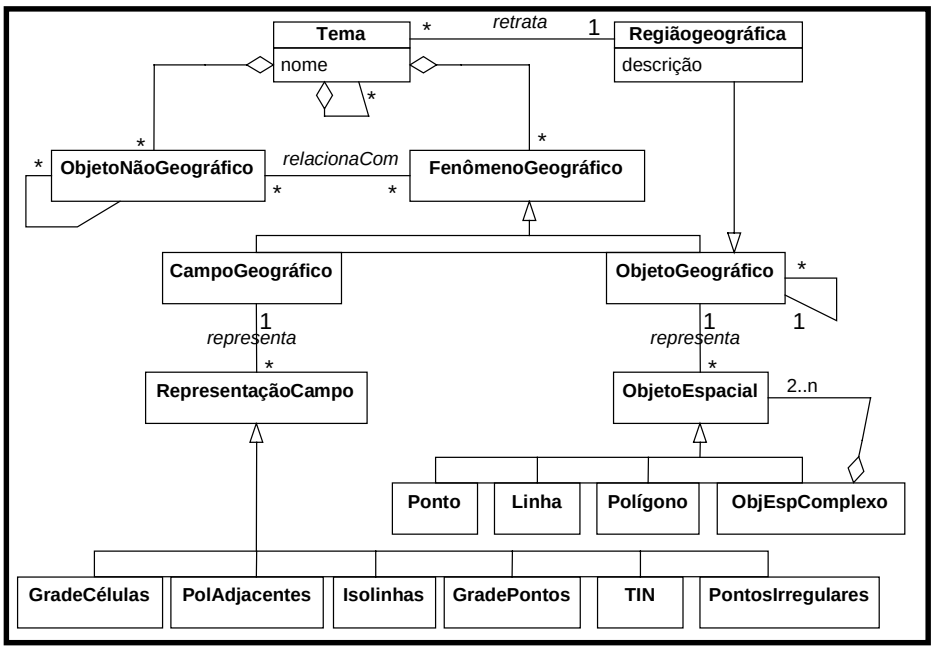


FIGURA 2.2 – Diagrama de classes do GeoFrame [LIS 2000a]

O *framework* é baseado nos conceitos das classes *Tema* e *RegiãoGeográfica*, as quais formam a base de qualquer aplicação geográfica. Toda aplicação geográfica tem como

objetivo o gerenciamento e a manipulação de um conjunto de dados para uma determinada região de interesse, constituindo um banco de dados geográficos.

Em um banco de dados geográficos, objetos que não possuem referência com relação a uma posição geográfica são classificados como *ObjetoNãoGeográfico*. A classe *FenômenoGeográfico* é uma classe abstrata, que generaliza qualquer fenômeno cuja localização em relação à superfície terrestre seja considerada.

As classes *CampoGeográfico* e *ObjetoGeográfico* são especializações da classe *FenômenoGeográfico,* permitindo ao projetista especificar, de forma distinta, porém integrada, os campos e os objetos geográficos, respectivamente.

A subclasse *ObjetoEspacial* é especializada nas subclasses ponto, polígono, linha e objeto espacial complexo. A subclasse *RepresentaçãoCampo* pode ser resumida em seis subclasses: grade de células; polígonos adjacentes; isolinhas; grade de pontos; TIN; pontos irregulares. No GeoFrame, todas as subclasses acima são tratadas como formas de abstração do componente espacial dos fenômenos geográficos, e não como serão armazenadas no banco de dados.

Esquemas conceituais são, então, elaborados a partir da especificação das classes do GeoFrame. Esquemas podem se valer de uma notação sucinta baseada nos esteriótipos ilustrados na Figura 2.3.

FIGURA 2.3 –Esteriótipos do *framework* GeoFrame [LIS 2000a]

A modelagem conceitual de banco de dados geográficos, usando o GeoFrame, é composta de três etapas [LIS 99]. Inicialmente, são identificados os diversos temas a serem projetados. Na segunda etapa, é definido um subesquema de classes para cada tema identificado. Ainda nessa etapa, é realizada a especificação das associações entre classes de diferentes temas. Por último, é identificado o tipo de representação que cada entidade geográfica identificada deve assumir no banco de dados, conforme a Figura 2.3.

Na Figura 2.3, o primeiro conjunto de esteriótipos é usado para diferenciar os três principais tipos de entidades pertencentes a um banco de dados geográficos (objeto não geográfico, objeto e campo). O GeoFrame apresenta um conjunto fixo de alternativas de representação geométrica, usando uma simbologia que distingue os objetos geográficos (segundo conjunto da Figura 2.3) e campos geográficos (terceiro conjunto da Figura 2.3). Nos esquemas conceituais, esses esteriótipos são utilizados para substituir as especializações das classes do modelo, simplificando significativamente o diagrama final.

Por último, o estereótipo <<função>> é usado para caracterizar um tipo especial de associação que ocorre quando da modelagem de campos categóricos. Segundo Chrisman [CHR 97], numa estrutura de cobertura categórica o espaço é classificado em categorias mutuamente exclusivas, ou seja, uma variável possui um valor do tipo categoria em todos os pontos dentro de uma região (ex.: tipos de solos).

Um esquema conceitual, baseado na abordagem UML-GeoFrame pode ser traduzido para a linguagem *SAIFTalk*, parte do padrão *SAIF*. Esse padrão e a sua linguagem estão descritos na próxima Seção.

2.2.2 Padrão de Intercâmbio e Armazenamento de dados - *SAIF*

A seleção do padrão *SAIF* (*Spatial Archive and Interchange Format*) para a utilização nesta pesquisa, está relacionada diretamente ao fato de ser um modelo de intercâmbio de dados espaciais, utilizado por uma ferramenta de tradução de dados espaciais que está em constante desenvolvimento [FME 2000].

O padrão apresenta duas características básicas que o torna indispensável neste trabalho. A primeira relaciona-se à representação das características espaciais das entidades da realidade. A segunda característica refere-se à estrutura do *SAIF*, a qual adota uma linguagem textual formal para representar cada classe, sendo essa linguagem interpretada por uma ferramenta de tradução de dados espaciais.

A ferramenta de tradução de dados utilizada nesta pesquisa também permite a utilização de outro padrão, denominado *SDTS* (*Spatial Data Transfer Standard*), que foi desenvolvido pelo *FGDG* (*Federal Geographic Data Committee*) dos Estados Unidos [MOE 88] e publicado no *US Federal Information Processing Standard* em 1992. Entretanto, por não especificar uma linguagem formal e por ser um padrão voltado para a transferência de metadados, não é utilizado neste trabalho. Dessa forma, a opção pela utilização do padrão *SAIF* é mais adequada para o desenvolvimento do processo.

Uma visão geral do padrão *SAIF* torna-se necessária, considerando que a linguagem formal *SAIFTalk* descreve os diagramas de classes desse padrão. Sendo assim, busca-se apresentar uma visão ampla da estrutura do padrão, descrito na Seção 2.2.2.1. A Seção 2.2.2.2 destina-se à definição da linguagem utilizada para descrever a hierarquia de classes definida pelo padrão *SAIF*, denominada *SAIFTalk* [SAI 95]. Esclarecimentos sobre esse padrão são apresentados na documentação do *SAIF* [SAI 95] e em [COS 2000].

2.2.2.1 Visão Geral da Estrutura do Padrão SAIF

O padrão *SAIF* define uma hierarquia de classes que oferece estruturas de dados, independente de software, para descrever dados geográficos em um nível mais abstrato que os padrões de dados geográficos em geral. O *SAIF* é proposto pelo *Survey and Resource Mapping Branch Ministry of Environment Lands and Parks* (MELP) e utiliza os conceitos do paradigma da O.O. e a notação gráfica OMT (*Object Modelling Technique*).

A estrutura do padrão *SAIF* divide-se em três níveis distintos. O primeiro nível é onde toda a informação é definida como construções matemáticas, incluindo coleções,

enumerações e primitivas. O segundo descreve os conceitos de construções espaciais e temporais. No último nível do padrão *SAIF*, torna-se possível representar entidades do mundo real como tipos de objeto geográfico, existindo no tempo e no espaço e com vários tipos de relacionamentos com outros objetos. Essa estrutura está ilustrada na Figura 2.4.

O construtor espacial e temporal, no centro da Figura 2.4, constitui o *SAIF Standard Schema*. Esse padrão é constituído de mais de 300 classes básicas, as quais são descritas em [SAI 95]. Abaixo são descritos os níveis: *data model*, *standard schema* e *user schema*.

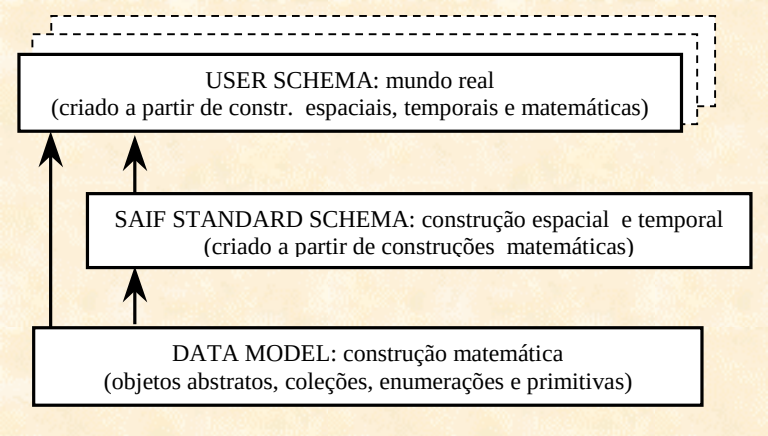


FIGURA 2.4 – Paradigma de modelagem *SAIF* [SAI 95]

- *Data Model*

Um modelo de dados é definido como um conjunto de conceitos e regras utilizadas para descrever tipos e relacionamentos entre objetos. O modelo de dados *SAIF* segue a O.O. e está organizado em uma hierarquia de tipo *(type)*, conforme ilustrado na Figura 2.5, que fornece semântica matemática, mas sem significado geométrico.

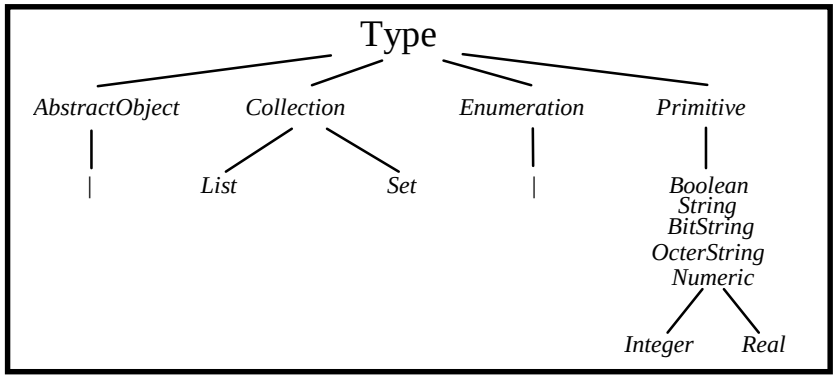


FIGURA 2.5 – Modelo de dados *SAIF* [SAI 95]

- *AbstractObject* - o tipo *AbstractObject* é a base do padrão, sendo as demais classes derivadas deste tipo. O *Data Model* não inclui nenhum tipo particular de *AbstractObject*, como ilustra a Figura 2.5, mas inclui o conceito de *subclassing*, indicando que a classe pode ser especializada pelo usuário, uma vez que, nem todas as superclasses do padrão podem ser estendidas, conforme descrito na Seção 2.2.2.2.

- *collection* - consiste em um grupo de valores de dados. Não podem ser instanciados diretamente, só através de listas (*list* - os valores têm uma ordem especificada e podem não ser únicos) ou conjuntos (*set* – não são ordenados e são obrigatoriamente únicos).

- *enumeration* - o tipo *Enumeration* é indicado através da construção *{enum}* ao lado do nome da classe. Quando a indicação *{enum}* estiver presente, torna-se possível enumerar um conjunto de descrições para essa classe. Por exemplo, cores primárias, que é um tipo de *enumeration*, pode apresentar algumas descrições como: verde, vermelho e azul.

- *primitive* - objetos deste tipo são dados simples utilizados como blocos básicos de construtores de valores de dados. Esse tipo está organizado em: *String*, *OctetString* (string de bytes), *BitString*, *Boolean* e *Numeric*. Este último oferece, ainda, duas hierarquias de tipos de dados, *Integer* e *Real*.

O *Data Model* é a base para o desenvolvimento adicional de classes e enumerações. Todas as classes do *Standard Schema* ou do *User Schema* são subclasses de *AbstractObject* ou de suas subclasses de *AbstractObject.* Da mesma forma, as enumerações são subclasses de *Enumeration*. Assim, as classes *AbstractObject* e *Enumeration* podem ser especializadas pelos usuários, porém, as classes *Collection* e *Primitive* não podem ser estendidas, são usadas apenas como domínio dos atributos, conforme ilustra a Figura 2.5.

➢ *Standard Schema*

O esquema do padrão *SAIF* define representações de fenômenos do mundo real, espaciais e temporais [SAI 95]. Representações do mundo real são as próprias entidades geográficas (ruas, vazamento de óleo, altimetria), as quais podem estar relacionadas entre si, incluindo relacionamentos espaciais e temporais como, por exemplo, um vazamento de óleo pode ocorrer durante uma grande tempestade.

O padrão fornece um grande número de classes espaciais e espaço-temporais, que são a base para o usuário definir novas classes (*User Schemas*). Em um nível mais elevado, o padrão *SAIF* define um conjunto de classes que compõem o *Standard Schema* (Figura 2.6).

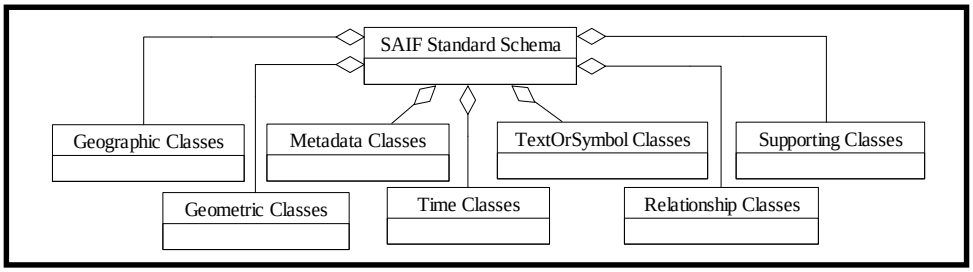


Figura 2.6 - Classes do *SAIF Standard Schema*

- *Geographic*: as classes desta hierarquia definem conceitos de objeto geográfico, espaço, tempo;

- *Geometric*: fornece um conjunto de classes para definir a geometria dos objetos geográficos;

- *Metadata*: proporciona meios para o usuário descrever os metadados;

- *Time*: oferece um conjunto de classes que expressam conceitos relacionados ao tempo como: data e período;

- *TextOrSymbol*: estabelece um conjunto de classes para definir tipos de textos e símbolos;

- *Relationship*: fornece um conjunto de classes que expressam relacionamentos espaço-temporais entre objetos;

- *Supporting*: define um conjunto de classes de suporte para as classes das outras hierarquias.

Conforme ilustrado na Figura 2.6, o esquema padrão *SAIF* é um agregado de várias classes. A apresentação exaustiva das classes está fora do âmbito deste trabalho, já que pode ser encontrada em [COS 2000]. Parte da hierarquia de *Geographic Classes* e *Geometric Classes* são descritas abaixo, visto que, alguns dos conceitos das classes que compõem essa hierarquia são correspondentes aos conceitos das classes do *framework* conceitual GeoFrame.

a) Geographic Classes: esta hierarquia representa o conceito central do *Standard Schema*, onde encontram-se definições de objetos geográficos, conceitos de espaço e tempo, bem como, anotações. As classes são ilustradas na Figura 2.7 e descritas abaixo.

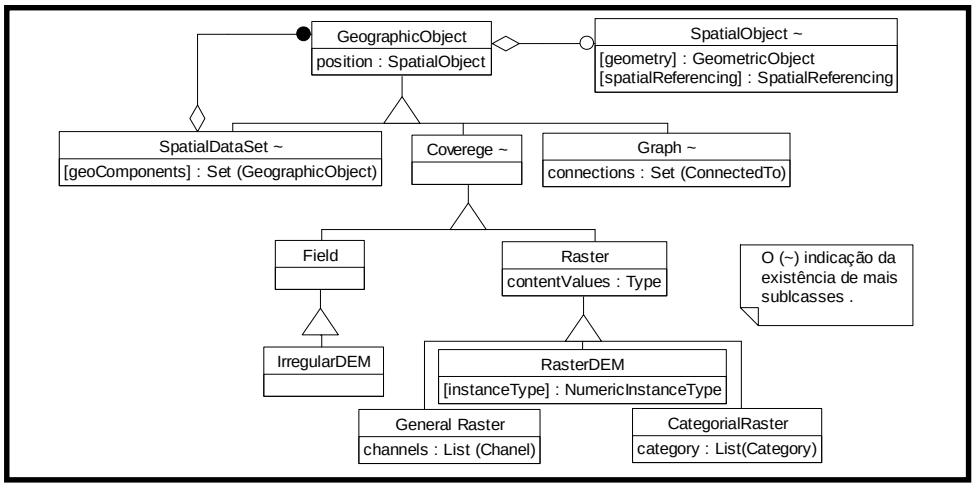


FIGURA 2.7 – Estrutura da *Geographic Classes*

a.1) GeographicObject - esta classe pode ser especializada pelo usuário, para descrever objetos geográficos, que representam entidades[2] e/ou fenômenos[3] do

mundo real, com uma posição em relação à superfície terrestre e/ou em determinado tempo no espaço. A posição de cada objeto no espaço é representada pelo atributo *position*, cujo domínio é espacial, classe *SpatialObject* definida no item (*b.1*).

a.2) SpatialDataSet - esta classe representa um conjunto de dados referenciados geograficamente, que são agrupados, por questões de similaridade ou complementaridade(referência espacial ou metadados). Um objeto pertencente a essa classe contém outros objetos geográficos, ou seja, é um conjunto de dados espaciais que contém subconjuntos de dados.

Um objeto, pertencente a qualquer subclasse de *SpatialDataSet*, conterá atributos da classe *SpatialObject* (item *b.1*), incorporando a geometria e o referenciamento espacial como seus atributos.

a.3) C*overages* - podem representar fenômenos geográficos distribuídos em uma determinada região do espaço como, por exemplo, imagens de satélite com pressão barométrica, descrição do uso da terra, etc. Conforme ilustrado na Figura 2.7, a classe *Coverage* é especializada em *Field e Raster*.

a.4) 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. Essa 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.5) Raster - é utilizada para descrever dados *raster* típicos, onde os valores são associados a cada posição em um *grid* (grade). A grade é definida em termos de estruturas regulares, multi-dimensional, retangular, estrutura hexagonal ou triangular. Essa subclasse pode ser especializada em três subclasses:

- *generalRaster*: é o uso típico da classe *raster*, envolvendo um grupo de uma ou duas dimensões de grade, sendo mais adequada em análise de imagem;
- *categoricalRaster*: descreve dados *raster*, onde cada valor da célula é correlacionada a uma categoria. Cada categoria possui uma única combinação de atributos, que é correlacionada a um conjunto de códigos, definidos pelo usuário.
- *rasterDEM*: define uma estrutura de grade, onde cada célula contém um valor em x, armazenada por um *octetString* ou uma referência a um arquivo externo.

a.6) Graph - consiste em uma série de objetos geográficos, que são conectados com outros objetos, implicando, necessariamente, que os objetos geográficos sejam contínuos no espaço ou no tempo.

---

[2] objetos discretos que têm limites ou extensões espaciais definidos (rios, edifícios. estradas).

[3] variam sobre o espaço e não têm uma extensão específica (temperatura, tipos de solo).

b) Geometric Classes: esta hierarquia agrupa objetos geográficos que são referenciados e representados geograficamente através de abstrações matemáticas como, por exemplo, pontos, linhas e polígonos. A seguir, na Figura 2.8 é ilustrada essa hierarquia de classes.

b.1) SpatialObject - todo objeto geográfico ocupa uma posição numa determinada região no espaço. Essa região pode ou não ser representada por um objeto espacial, conforme ilustra a Figura 2.8. Existem dois aspectos de um objeto espacial a considerar, a geometria (*geometry*) e a referência espacial (*spatial referencing*). A geometria representa a forma do objeto geográfico e a referência espacial define a localização de um objeto em um sistema de referência espacial.

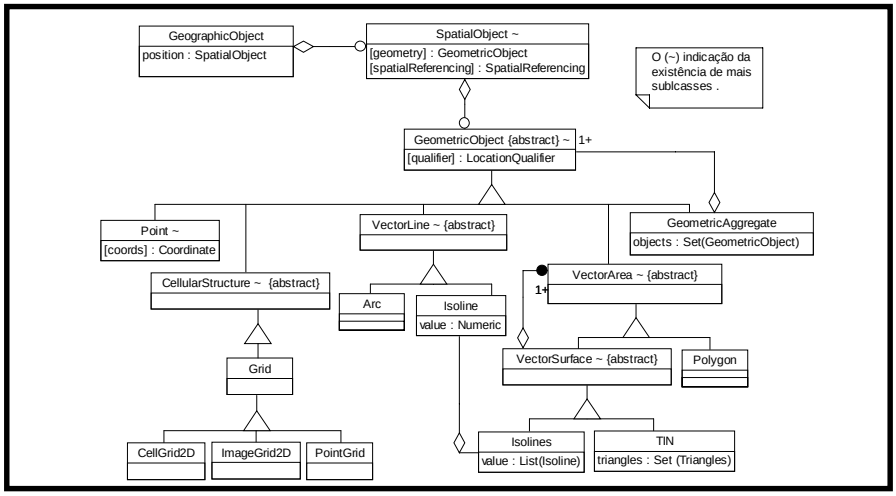


FIGURA 2.8 – Estrutura de *Geometric Classes*

b.2) GeometricObject - esta é uma classe abstrata, que generaliza as formas de representação espacial dos objetos georreferenciados. Os objetos pertencentes a essa classe tem posição espacial especificada através de seus componentes. Conforme ilustrado na Figura 2.8, a classe *GeometricObject* está especializada em *Point*, *CellularStructure*, *VectorLine*, *VectorArea* e *GeometricAggregate*.

b.3) Point: é utilizada para descrever a posição de um simples ponto na superfície terrestre.

b.4) 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. A Figura 2.8 ilustra essa estrutura.

- *CellGrid2D*: esta classe é instanciada por objetos no formato *raster*, sendo estes bidimensionais.
- *ImageGrid2D*: define o espaço ocupado por imagens bidimensionais.
- *PointGrid*: esta classe é apropriada para definir grades de *n* dimensões de pontos.

b.5) VectorLine: representa objetos geométricos com uma dimensão, sendo definido espacialmente através de coordenadas. Sua estrutura é ilustrada na Figura 2.8.

- *Arc:* conjunto de dois ou mais pontos conectados a uma lista, que define a posição dos pontos.
- *Isoline:* cada isolinha é uma parte do conjunto que forma as *isolines* (especificado em *VectorArea*). O atributo *value* indica o valor específico de cada isolinha.

b.6) VectorArea: representa objetos geométricos de duas ou três dimensões (*VectorSurface*) no formato de vetor. A Figura 2.8 ilustra a estrutura dessa classe.

- *Polygon*: esta classe é utilizada para especificar polígonos primários. Polígono é uma figura plana formada por uma linha poligonal fechada.
- *VectorSuface*: é uma subclasse especial de *VectorArea.* Essa classe é utilizada se a área do objeto necessitar de coordenadas tridimensionais.
  - *Isolines*: esta estrutura é formada por uma lista de *isoline*.
  - *TIN:* estrutura de dados que descreve uma superfície através de uma rede de triângulos irregulares.

b.7) GeometricAggregate: representa a agregação de várias representações de objetos geométricos (Figura 2.8).

➢ *User Schema*

Neste nível, o usuário pode criar um esquema com novos *ADTs* (*Abstract Data Type*), para descrever a semântica das entidades e dos fenômenos do mundo real. Os *ADTs* podem ser construídos, utilizando-se, como base, as classes construtoras espaciais, temporais e espaço-temporais do *Standard Schema* e as classes matemáticas do *Data Model*.

As entidades e/ou fenômenos definidos pelo usuário são modelados como subclasses do *Standard Schema*. Porém, nem todas as classes do *Standard Schema* permitem ser especializadas pelo usuário, conforme apresentado na Seção a seguir.

2.2.2.2 DEFINIÇÃO DA LINGUAGEM SAIFTALK

Como referido anteriormente, o padrão *SAIF* é um modelo de intercâmbio de dados espaciais e disponibiliza uma linguagem formal para descrever textualmente todas as classes que o compõem. A linguagem (*script*) *SAIFtalk* oferece essa estrutura formal para solucionar problemas de interpretação que ocorrem quando se pretende transferir informações. Entretanto, apesar de ser uma linguagem eficaz para o gerenciamento de dados, ela não é tão completa quanto algumas linguagens tradicionais [STR 98].

Esse *script* é formado por dois componentes: *Class Syntax Notation* (CSN) e *Object Syntax Notation* (OSN). Ambas as notações são utilizadas, respectivamente, para definição de classes e para instanciação de classes (objetos) e estão descritas abaixo.

➢ *Class Syntax Notation (CSN)*

Em [COS 2000] são apresentadas as sintaxes da linguagem CSN para a definição de classes, de enumerações e de domínios. Tendo em vista que o *framework* conceitual

GeoFrame não trabalha com o conceito de enumerações e que os domínios são de tipos primitivos, apenas a sintaxe da classe é apresentada neste trabalho.

A notação desse componente da linguagem é baseada na gramática BNF (*Forma Backus-Naur),* que foi estendida para atender às necessidades dos objetos espaciais. No Anexo 1, essa gramática é descrita com a aplicação do componente CSN.

A sintaxe da linguagem, para a especificação de classes, inicia com o par < > para indicar o início e o fim da classe que está sendo definida. Essa notação permite a utilização de aspas para documentar comentários a respeito da classe.

A notação CSN apresenta as cláusulas descritas no exemplo abaixo, sendo os dois primeiros itens obrigatórios e os demais, opcionais.

< Superclass::NameSchema1		
subclass:	*Subclass::NameSchema2*	
subclassing:	*"The user may define subclasses of this class."*	
attributes:	*attribute13*	*DomainForAttribute13*
defaults:	*attribute13:*	*valueW*
restricted:	*attribute1:*	*RestrictedDomainForAttribute1*
	attribute2:	*valueX*
constraints:	*"The product of the values for attribute3 and attribute4 must be greater than the value for attribute5."*	
comments:	*“This class definition serves as a template for other class definitions”.*	
>		

- *superclass* - corresponde à definição da superclasse. Se a superclasse não é parte do *Data Model* (*AbstractObject ou Enumeration*) ou do *Standard Schema*, o nome da superclasse é seguido por dois pontos e pelo nome do esquema externo (esquema do usuário) ao qual ela pertence. Caso a superclasse seja parte do padrão *SAIF*, os dois pontos e o nome do esquema são omitidos.

- *class Name* (subclasse) - especifica o nome da subclasse (*Standard Schema* ou *User Schema*), juntamente com o nome do esquema externo. Se a subclasse é parte do *Standard Schema*, o nome do esquema não é declarado. O nome das classes e o nome do esquema iniciam com letra maiúscula e podem ser seguidos por letras maiúsculas ou minúsculas, dígitos numéricos ou sublinhas (_).

- *subclassing* - utilizada para declarar se a superclasse do *SAIF Schema Standard* pode ser especializada pelo usuário, uma vez que, nem todas as superclasses do padrão podem ser estendidas.

- *attributes* - a função dos atributos é caracterizar os objetos de uma classe. Cada um deles apresenta um tipo de domínio específico, podendo ser *primitive, enumeration, collection* ou *class*. O valor de um atributo pode ser opcional ou obrigatório. Sendo obrigatório, ele deve ser especificado para cada objeto da classe a que pertence.

- *defaults* - um valor *default* é definido como o conteúdo inicial de um atributo. Se o atributo for opcional e o valor nulo não for considerado como conteúdo válido, é necessário definir um valor *default* para o atributo. Se ele for obrigatório, um

valor *default* não precisa, necessariamente, ser definido, pois não haverá a possibilidade de se inserir um valor nulo para o atributo.

- *restricted* - corresponde às definições que restringem ou limitam o domínio dos atributos. Um atributo pode ser limitado tanto por um tipo de domínio (*integer*), quanto por um intervalo de valores. Alternativamente, esta cláusula pode ser utilizada para definir domínios mais complexos como, por exemplo, limitar a geometria do objeto.

- *comments* - permite especificar comentários sobre as classes, os atributos e outras informações, e ainda, incluir diagramas, textos ou tabelas.

➢ *Object Syntax Notation (OSN)*

Como descrito no item anterior, a notação sintática dos componentes de *SAIFtalk* tem por base a gramática BNF. No Anexo 2, essa gramática é descrita, aplicada ao componente OSN.

A notação da sintaxe é embasada em dois conceitos:

- todas as classes, definidas no esquema *SAIF*, são derivadas diretamente ou através do mecanismo de herança, de *AbstractObject*, que é a superclasse abstrata do *Standard Schema*; e

- todas as instâncias das classes são definidas em termos de tipos *primitives* ou descrições de *enumerations*.

Essa notação inicia com a declaração do nome, seguido por um parênteses, dentro do qual são definidos os atributos da classe. Os atributos são declarados de forma ordenada: primeiro; os atributos herdados, e depois; os especificados pela classe.

Cada atributo é definido pelo seu domínio. Se o domínio é uma *enumeration* ou *primitive*, o valor é fornecido diretamente. Se o domínio é complexo, isto é, uma classe, seu nome é declarado seguido por parênteses, dentro dos quais são descritos seus atributos, recursivamente, até que todos os domínios sejam do tipo *primitive*.

Deve-se observar que, caso o domínio seja um tipo *list* ou um *set*, seus elementos são declarados entre chaves, conforme ilustra o exemplo abaixo. Caso o domínio seja uma *string*, ela deve vir entre aspas duplas.

```
Instance1 (
        position:SpatialObject (
            geometric:GeometricObjectType (
                coords:CoordinateType
            )
            spatialReferencing: SpatialReferencingType
        )
        attributte1 :  integerValue
        attribute 2:  Class10 (
            attributeA : True
        ) }
```

A relação de correspondência entre o esquema conceitual, gerado com base no *framework* GeoFrame, e a linguagem formal *SAIFTalk* que descreve o diagrama do padrão *SAIF* são demonstradas através de dois conjuntos de regras que são definidos no Capítulo 3.

2.3 Ferramentas de Suporte

Esta Seção tem por objetivo descrever as ferramentas necessárias para a elaboração de um ambiente de suporte que possibilite a validação das regras apresentadas no Capítulo 3. Desta forma, a Seção 2.3.1 descreve brevemente a ferramenta *RarionalRose2000*, a qual esta sendo utilizada neste trabalho, devido a algumas caracterísiticas essenciais como, por exemplo, a permissão para criar novos esteriótipos e disponibilização de um editor para gerar códigos. A Seção 2.3.2 apresenta a ferramenta de tradução de dados espaciais, denominada *FME*.

2.3.1 Ferramenta RationalRose2000

Esta ferramenta é um software para modelagem visual, que permite ao projetista criar, analisar, projetar, modificar e manipular seus componentes. Entende-se, neste caso, por modelagem visual, o processo de mapeamento do mundo real para um sistema de representação gráfica, através da utilização dos diagramas disponíveis nessa ferramenta [RAT 2000].

Existe uma variedade de diagramas que são parte da ferramenta. Cada item da lista abaixo representa um tipo de diagrama que compõe a ferramenta, sendo que uma mesma aplicação pode ser representada por múltiplos diagramas. Os diagramas da ferramenta são:

- diagrama de classes;
- diagrama de caso de uso;
- diagrama de colaboração;
- diagrama de estado;
- diagrama de sequência;
- diagrama de componente;
- diagrama de desenvolvimento;
- diagrama de atividade.

Neste trabalho, o destaque refere-se, especificamente, ao diagrama de classes, pois é nessa especificação que o *framework* conceitual GeoFrame modela os fenômenos da realidade a partir do mundo real, utilizando os conceitos da orientação a objetos e a linguagem UML, suportados pela ferramenta. Além dessas características disponíveis no *RationalRose2000*, cabe destacar mais algumas, que tornaram-se determinantes para

a seleção dessa ferramenta neste trabalho. Dentre os recursos da ferramenta, pode-se destacar como características dessa versão:

- configuração de menus;
- permissão para incluir informações adicionais ao modelo (esteriótipos), conforme definido pelo padrão UML;
- disponibilização de um editor de *script* (ambiente para criar, *debugger* e compilar os códigos) para geração de novos *scripts*.

Para melhor compreensão da ferramenta *RationalRose2000*, torna-se necessário apresentar, de forma sucinta, os seus componentes, bem como, a definição das partes que foram utilizadas para o desenvolvimento do processo de tradução GeoFrame-*SAIF*.

Basicamente, essa ferramenta apresenta um metamodelo, denominado REI (*Rational Rose Extensibility Interface Model*), no qual são definidas e controladas todas as funções e aplicações do *Rose* como, por exemplo, pacotes, classes, propriedades e métodos. A Figura 2.9 ilustra os componentes da ferramenta.

O metamodelo REI é definido como um conjunto comum de interface, usado pelo *Rational Rose Script* ou *Rational Rose Automation*, com a finalidade de acesso ao *Rose.*

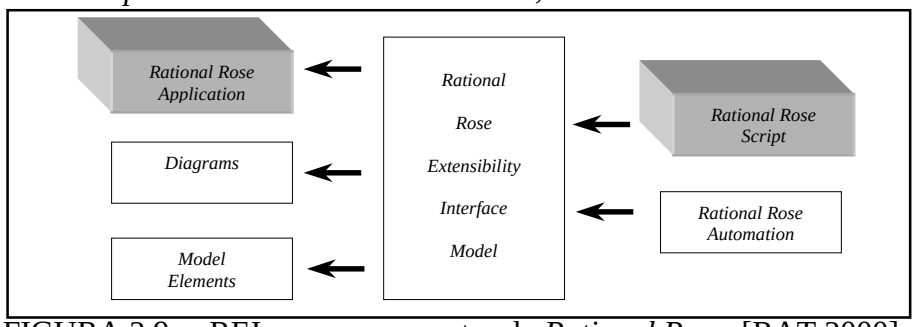


FIGURA 2.9 – REI e os componentes do *Rational Rose* [RAT 2000]

O conjunto de objetos do *Rose Script* permite automatizar as funcionalidades que não estão disponíveis no *Rose*, entendendo-se por "funcionalidades" a geração de códigos para a realização de uma determinada tarefa. Essas funcionalidades, bem como outros itens, são geradas pelos *Add-ins* (Capítulo 4) e permitidas pelos objetos do *Rose Script*.

O conjunto de objetos *Rose Automation* permite a integração entre algumas aplicações e o *Rose*, através de duas formas distintas. A primeira forma, utiliza o *Rose Automation* como um controlador automático, onde é possível inserir um objeto OLE dentro do *Rose. P*or exemplo, o *Rose* pode utilizar-se de um objeto OLE para executar alguma função de uma determinada aplicação (*word, excel*). A segunda utiliza o *Rose Automation* como um servidor automático, o qual permite chamar um objeto OLE dentro de outra aplicação OLE.

Os objetos dos componentes *Model Elements, Diagrams* e *Rose Application* controlam a interface dos elementos do modelo, dos diagramas e das visões, e das funcionalidades das aplicações, respectivamente. Assim sendo, as definições para estender a ferramenta *Rose*, encontram-se descritas nesses objetos. Os objetos do *Rose Application* tratam especificamente de interface para funcionalidades, logo, definem a interface do editor de *script* (*Rational Rose Script Editor*) para a geração de qualquer código.

Em destaque, na Figura 2.9, encontram-se os componentes que tratam das funcionalidades. Esses componentes permitem, entre outros itens, a automatização de tarefas que não estão presentes na ferramenta. Nesse caso, permitem a geração automática de arquivos *SAIF*, motivo pelo qual a ferramenta *Rose* é utilizada neste trabalho.

O arquivo resultante desse processo de mapeamento entre esquema conceitual e a linguagem *SAIFTalk*, ainda a nível conceitual, serve de entrada para um software de tradução automática de dados espaciais denominado *FME* que é apresentado na próxima Seção.

2.3.2 Ferramenta de Tradução de Dados Espaciais – *FME*

Historicamente, a tarefa de transformar dados de um formato para outro tem sido difícil. Como resultado, os usuários armazenam arquivos com grandes quantidades de dados, os quais estão restritos ao formato de uma única ferramenta de SIG. Inserida nesse contexto, a ferramenta *FME* (*Feature Manipulation Engine*) permite a tradução de modelos de dados, bem como a tradução dos seus dados de um software de SIG origem para outro destino.

A ferramenta *FME* consiste em módulos que executam traduções. Essas traduções são realizadas através de arquivos denominados *semantic mapping*, que podem ser selecionados pelo usuário durante a tradução. Esses arquivos especificam todos os aspectos necessários para a realização da tradução de um formato de arquivo origem para um destino. Alguns desses arquivos semânticos acompanham a ferramenta, tornando o processo mais eficiente.

O arquivo semântico contém um programa, especificando como o processo de tradução ocorre, juntamente com as regras de tradução específicas para cada software de SIG. Essas regras estão embutidas no módulo da ferramenta *FME* e definem como representar e descrever os elementos que estão contidos em um determinado software (origem) para outro software de SIG (destino). Esses arquivos semânticos são elaborados apenas uma vez, para cada dois softwares de SIG, em particular, e reutilizados a cada conjunto de dados traduzidos. Para melhor compreensão do funcionamento da ferramenta, torna-se necessário apresentar sua arquitetura como ilustra a Figura 2.10.

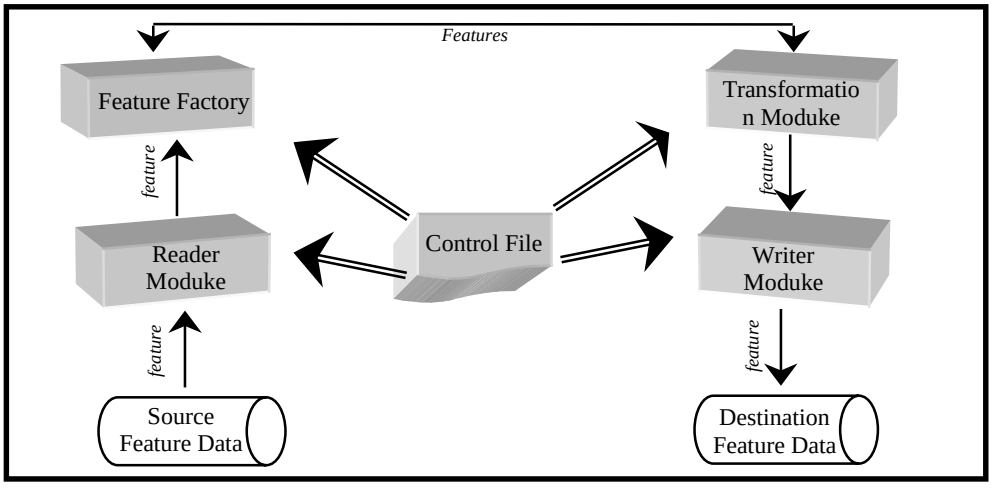


FIGURA 2.10 Arquitetura da ferramenta *FME* [FME 2000]

Como pode-se observar na Figura 2.10, a arquitetura é composta de vários módulos interligados. Um dos conceitos importantes contidos na ferramenta trata de *feature*, cuja definição é considerada como uma coleção de atributos e valores associados a uma geometria. Conforme descrito em [FME 2000], as *features* podem conter os seguintes tipos de geometria: ponto, linha, polígono, polígono com buracos, objeto agregado (coleção distinta de entidades geométricas, tratadas como uma única entidade).

Na Figura 2.10, o módulo *reader* é responsável pela recuperação de dados da origem. Os dados da origem podem ser arquivos individuais ou uma pasta com vários arquivos. O módulo *factory* processa cada feição e verifica a sua estrutura, dependendo da análise obtida, esse módulo separa ou une as feições, utilizando regras pré-definidas internamente pela ferramenta. O módulo *transformation* é responsável pela tradução de feições de um formato de arquivo para outro e é onde se encontram os arquivos *semantic mapping*. O módulo *writer* exporta as feições da ferramenta *FME* para o destino selecionado durante a tradução.

Atualmente, a ferramenta *FME* conta em média com 45 tipos de arquivos ou sistemas de SIG disponíveis para a realização da tradução de modelos e de dados espaciais. Entre essa lista, destacam-se o *AutoCad, Arc/Info, Esri Shape, Intergraph MGE, OracleSpatial* e o padrão *SAIF*. A lista dos arquivos e softwares de SIG suportados pela ferramenta *FME*, bem como os próximos arquivos que farão parte da mesma encontram-se descritos em [FME 2000].

O *SAIF* é conhecido como um padrão de armazenamento e intercâmbio de dados espaciais e pode ser traduzido para a maioria dos arquivos ou sistemas de SIG disponíveis na biblioteca interna do *FME*. Além dessa vantagem, acrescenta-se que a ferramenta está em constante atualização, logo, aumentando progressivamente o número de softwares suportados.

Outra vantagem relaciona-se à disponibilidade de um arquivo *semantic mapping* para *SAIF*, o qual é capaz de ler, interpretar o arquivo *SAIF* e mapeá-lo para vários formatos de SIG, desde que disponíveis na biblioteca do *FME*.

O arquivo semântico *SAIF* é apenas um dos vários tipos de arquivos oferecidos pela ferramenta *FME*. Também existe a possibilidade de elaborar um arquivo semântico manualmente, através da estrutura interna disponível na ferramenta.

A lógica de funcionamento da ferramenta *FME*, de certa forma, é semelhante à proposta neste trabalho. No *FME*, as regras de mapeamento são elaboradas a nível de modelos de dados lógicos de um software de SIG para outro, sendo que o objetivo da pesquisa é elaborar regras para mapear diagramas de classes do nível conceitual para o nível lógico de alguns softwares de SIG.

Para atender o objetivo da pesquisa, busca-se mapear os esquemas conceituais, baseados no *framework* GeoFrame, para a linguagem estruturada *SAIFTalk* do padrão *SAIF*, conforme descrito no próximo Capítulo.

3 Mapeamento de Esquemas Conceituais para o Padrão *SAIF*

Como descrito na Seção 1.2, um dos pontos de interesse da pesquisa é aprimorar o *framework* GeoFrame [LIS 99] como uma ferramenta de modelagem conceitual de SIG. Nesse contexto, este Capítulo apresenta dois conjuntos de regras elaborados para realizar o mapeamento de esquemas conceituais, baseados no GeoFrame, para a linguagem formal do padrão *SAIF*, que é utilizada neste trabalho como intermediária do processo de mapeamento, para alcançar o objetivo proposto.

O padrão empregado para esse mapeamento deve-se, principalmente, ao fato dele permitir a representação das características espaciais consideradas relevantes na realidade geográfica e disponibilizar uma linguagem formal, para definir a especificação dos diagramas de classes que retratam a realidade, a qual pode ser traduzida para os modelos lógicos de alguns SIG.

Embora o padrão *SAIF* apresente as características citadas acima, as quais o tornaram um caminho alternativo para concretizar o mapeamento entre os esquemas conceituais e os esquemas lógicos de alguns softwares SIG, através de sua linguagem, não é possível empregá-lo diretamente como base para o desenvolvimento de esquemas conceituais devido a dois aspectos básicos:

- a inexistência de integração entre a dicotomia da visão de campo e objeto, a qual é explorada pelo *framework* GeoFrame e é considerada de suma importância para a modelagem de dados espaciais;

- a complexidade do diagrama de classes oferecido pelo padrão *SAIF*, o qual foi elaborado para tratar o intercâmbio de dados espaciais em vários níveis de abstração, tornando-se um modelo que apresenta muitos detalhes das características computacionais dos dados, prejudicando a visão abstrata que o projetista deve possuir quando modela os objetos do mundo real.

Portanto, considerando os aspectos citados acima, verifica-se o *framework* conceitual GeoFrame é mais adequado para a modelagem de esquemas conceituais do que o padrão *SAIF*, devido a existência da dicotomia entre a visão de campo e de objetos, elevando o seu nível de abstração; além de produzir esquemas mais compactos que os outros modelos, devido ao conteúdo semântico de suas primitivas de modelagem.

Como o GeoFrame não disponibiliza uma linguagem para interpretar o seu diagrama de classes, optou-se pela utilização de algumas cláusulas oferecidas pela linguagem do padrão *SAIF*, sendo essa empregada como intermediária no processo de mapeamento de esquemas conceituais, baseados no GeoFrame, para os esquemas lógicos de alguns softwares de SIG.

Dessa forma, as regras de mapeamento, descritas neste Capítulo, buscam traduzir qualquer esquema conceitual, baseado no *framework* GeoFrame, para a linguagem formal *SAIFTalk*, adotada pelo padrão *SAIF*.

3.1 Definição de Regras de Mapeamento

O padrão *SAIF*, descrito na Seção 2.2.2, apresenta uma hierarquia de classes abrangente, possibilitando assim, uma representação das características espaciais consideradas relevantes para a modelagem da realidade geográfica.

Cabe destacar que, a primeira aproximação de um conjunto de regras entre o *framework* GeoFrame e o padrão *SAIF* deve-se a uma análise comparativa dos diagramas de classes de ambos. Posteriormente, parte-se para a investigação da linguagem formal *SAIFTalk*, definida para representar textualmente os diagramas de classes do padrão *SAIF*.

Para a definição de regras genéricas de mapeamento, considera-se dois conjuntos de regras. A Seção 3.1.1 apresenta o primeiro conjunto, que define regras de correspondência, determinando uma relação entre os conceitos da realidade percebidos pelo diagrama de classes básicas do GeoFrame e a linguagem formal apresentada pelo padrão *SAIF*. A Seção 3.1.2 trata de regras de transformação, onde procura-se mapear os conceitos do paradigma da orientação a objetos, empregados pelo *framework* conceitual, para os conceitos relacionais empregados pela maioria dos softwares de SIG, disponíveis atualmente no mercado.

3.1.1 Regras de Correspondência

As regras de correspondência buscam converter o diagrama do GeoFrame em linguagem do padrão *SAIF*.

Conforme a descrição do diagrama de classes do GeoFrame (Seção 2.2.1) e da hierarquia de classes do padrão *SAIF* (Seção 2.2.2), torna-se possível perceber, através da análise, que a origem dos conceitos do diagrama de classes do GeoFrame encontra-se em determinadas classes que formam os componentes *GeographicObject* e *GeometricObject* do padrão *SAIF*. Contudo, essas classes são modeladas no GeoFrame de forma diferenciada do padrão *SAIF*.

Nesse aspecto, o projetista que modela o esquema de classes a partir do GeoFrame, utiliza-se de esteriótipos (Figura 2.3), que permitem diferenciar na modelagem, tanto os tipos de entidades (objeto não geográfico e fenômenos), quanto a representação espacial desses fenômenos geográficos, tornando assim, o modelo menos sobrecarregado. O mesmo não acontece com a modelagem do padrão *SAIF*, na qual o projetista deve especializar as classes da aplicação (objetos, campos, objetos não geográficos, tema) a partir especialização da estrutura da classe *GeographicObject* do *SAIF*. A representação espacial, dependendo do objeto, pode estar ou não associada a alguma das subclasses geométricas definidas pela estrutura da classe *GeometricObject* do modelo *SAIF*, conforme ilustra a Figura 2.8.

Dessa forma, os componentes da classe *objetoNãoGeográfico, fenômenoGeográfico, campoGeográfico, objetoGeográfico* e *tema* do GeoFrame correspondem a conceitos da classe *GeographicObject* do padrão *SAIF*. A mesma correspondência ocorre entre as classes *representaçãoCampo, objetoEspacial* e suas respectivas subclasses, definidas no diagrama do GeoFrame e os componentes da classe *GeometricObject* do *SAIF*.

No entanto, existe um conceito utilizado pelo GeoFrame que não apresenta correspondência no padrão *SAIF*. Trata-se da classe *regiãoGeográfica, esse* termo como área de estudo, é utilizado em muitos SIG, porém, refere-se a valores definidos, em muitos casos, implicitamente pelos dados introduzidos nos SIG durante a inserção dos dados na aplicação. A inexistência de correspondência do conceito de *regiãoGeográfica* entre o *framework* conceitual GeoFrame e o padrão *SAIF*, não diminui a qualidade do resultado final do mapeamento, visto que esse conceito não se encontra presente nos modelos lógicos dos softwares de SIG comerciais.

A definição de um esquema genérico de mapeamento entre o *framework* conceitual GeoFrame e as classes do padrão *SAIF* baseia-se na estrutura de um fluxograma, ilustrado na Figura 3.1.

De acordo com a Figura 3.1, nota-se que as entidades podem estar agrupadas em um pacote, conceito definido pela UML, ou não. No primeiro caso, esse conceito denomina-se *tema* e pode ser representado no *SAIF* através da classe *SpatialDataSet*. No segundo caso, verifica-se que as entidades do mundo real podem ser modeladas como *objetoNãoGeográfico* ou *fenômenoGeográfico*. A primeira classe corresponde aos conceitos da classe *geographicObject* do *SAIF*, porém com a cláusula *restricted*, indicando que a classe *geographicObject* não deve apresentar associação com a classe *SpatialObject*, demonstrada no fluxograma da Figura 3.1, através da expressão "*position.geometry:nul*". A segunda classe, *fenômenoGeográfico*, está representada no fluxograma através de suas especializações, *objetoGeográfico* e *campoGeográfico.*

Os fenômenos modelados como *objetoGeográfico* no GeoFrame, correspondem à especializações da classe *GeographicObject* do *SAIF*. Caso contrário, se são modeladas como *campoGeográfico*, tornando-se especializações das subclasses de *Coverage* (item *a3*).

Conforme ilustrado na Figura 3.1, as classes *objetoEspacial* e *representaçãoCampo* do *framework* especializam um conjunto mínimo de construtores para a especificação do componente espacial dos fenômenos geográficos. Essa representação é ilustrada no fluxograma, através das suas subclasses *(ponto, linha, polígono, objEspComplexo, redeTriangularIrregular, pontosIrregulares, gradePontos, polígonoAdjacente, isolinhas, gradeCélulas).* No *SAIF*, as representações dos fenômenos espaciais são definidas como subclasses de *GeometricObject,* a qual especializa-se em um vasto conjunto de classes que expressam conceitos semelhantes aos dos construtores espaciais do GeoFrame. As subclasses do *SAIF* correspondentes às subclasses definidas para representação espacial do GeoFrame são *point, arc, polygon, geometricAggregate, TIN, DEMpoint, pointGrid, cellGrid2D, isolines* e *imageGrid2D.*

O fluxograma, ilustrado na Figura 3.1, apresenta de forma genérica a correspondência entre as classes do padrão *SAIF* e as classes do GeoFrame**.**

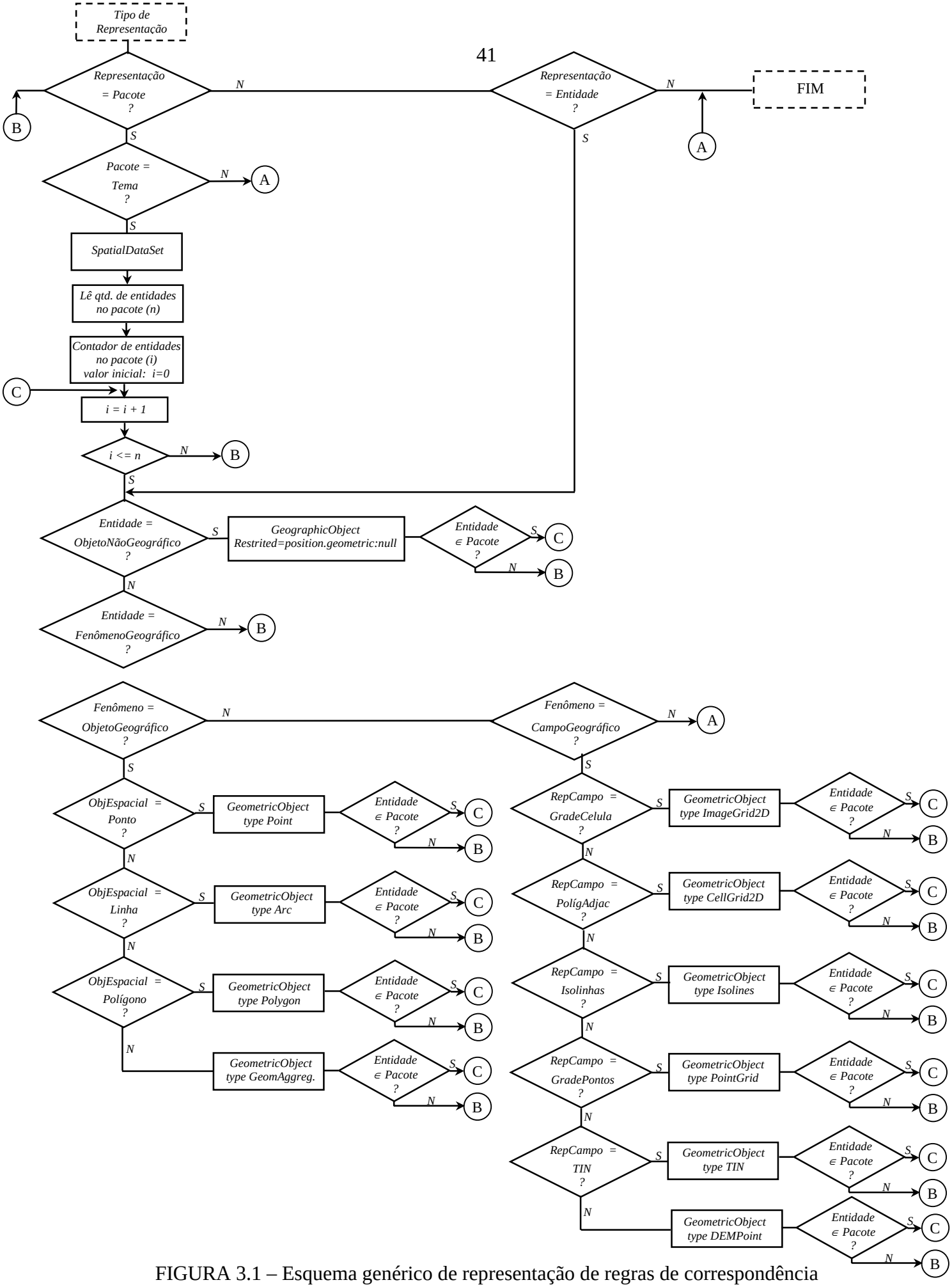
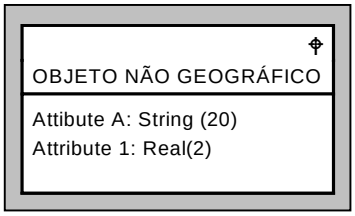


FIGURA 3.1 – Esquema genérico de representação de regras de correspondência

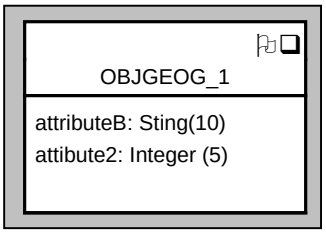
Outra forma de representar um esquema genérico para mapeamento de esquemas conceituais, baseados no GeoFrame, para a linguagem *SAIFTalk*, é através da própria linguagem *SAIFTalk*, como ilustra o exemplo abaixo. Cabe destacar que a cláusula *restricted*, descrita na Seção *Class Syntax Notation*, representa a forma do componente espacial da entidade geográfica. Outra cláusula presente nos *scripts* desse trabalho, trata-se da indicação *MySchema*, essa expressão indica que a classe pertence ao esquema elaborado pelo usuário, conforme descrito na Seção 2.2.2.2.

Um *objeto não geográfico* no GeoFrame é traduzido para a linguagem do padrão *SAIF*, como uma subclasse de *GeographicObject*, sem representação espacial, que é representada pela cláusula *restricted*, como ilustra o exemplo, abaixo:

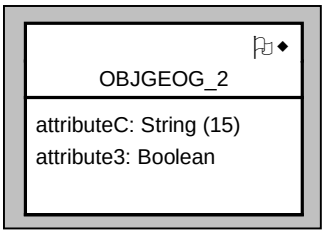



```
<Geographic Object
  subclass:     ObjetoNaoGeografico::MySchema
  attributtes:  attributeA  String(20)
                attribute1  Integer(2)
  restricted:   position.geometry:null
  comments:     comments
 >
```

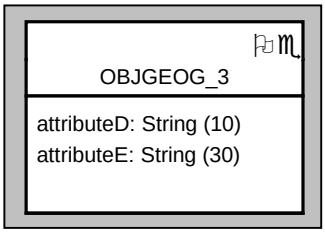
Um *objeto geográfico* do GeoFrame com a representação espacial do tipo *ponto, linha, polígono ou objeto espacial complexo* é representado no *SAIFTalk* como uma subclasse de *GeographicObject*, associada, respectivamente, à classe *point*, *arc*, *polygon* ou *geometricAggregatte*. Essa associação é representada através da cláusula *restricted.*



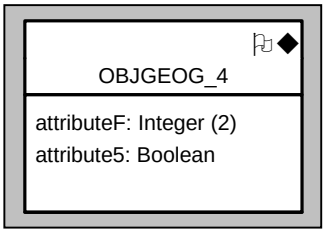

```
<Geographic Object
  subclass:     ObjGeog_1::MySchema
  attributtes:  attributeB      String(10)
                attribute2      Integer(5)
  restricted:   position.geometry:point
  comments:     comments
 >
```




```
<Geographic Object
  subclass:     ObjGeog_2::MySchema
  attributtes:  attributeC      String(15)
                attribute3      Boolean
  restricted:   position.geometry:arc
  comments:     comments
 >
```



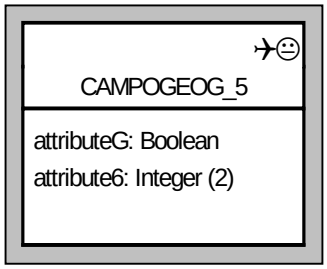

```
<Geographic Object
  subclass:     ObjGeog_3::MySchema
  attributtes:  attributeD      String(10)
                attributeE      String(30)
  restricted:   position.geometry:polygon
  comments:     comments
 >
```




```
<Geographic Object
  subclass:     ObjGeog_4::MySchema
  attributtes:  attributeF      Integer(2)
                attribute5      Boolean
  restricted:   position.geometry:geomAgg
  comments:     comments
 >
```

Todas as classes da visão de campo do GeoFrame são descritas em linguagem formal, utilizando-se da hierarquia completa definida pelo diagrama de classes do *SAIF*, conforme descrito abaixo.

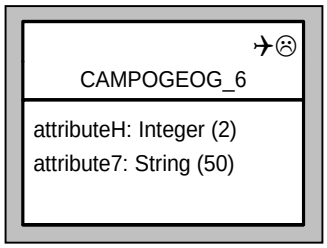
Um *campo geográfico* com representação espacial do tipo *grade de célula* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *GeneralRaster* associada à classe *ImageGrid2D* do *SAIF*.




```
<Geographic Object
      subclass:  Coverage
      comments:  comments
>
<Coverage
      subclass:  Raster
      comments:  comments
>
```

```
<Raster
     subclass:    GeneralRaster
     attributes:  [indexScheme]
     restricted:  position.geometry:Grid
     comments:    comments
>
<GeneralRaster
       subclass:   CampoGeogr_5::MySchema
       attributes: attributeG   Boolean
                   attribute6   Integer(2)
       restricted: position.geometry:ImageGrid2D
       comments:   comments
>
```

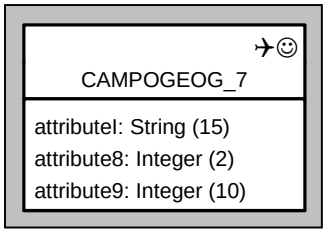
Um *campo geográfico* com representação espacial do tipo *polígono adjacente* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *CategorialRaster* associada à classe *CellGrid2D* do *SAIF*.




```
<Geographic Object
      subclass:  Coverage
      comments:  comments
>
<Coverage
      subclass:  Raster
      comments:  comments
>
```

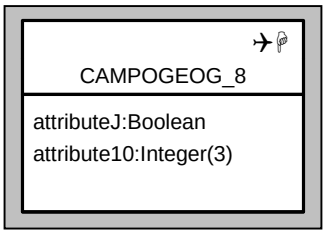
```
<Raster
      subclass: CategoricalRaster
      attributes:       [indexScheme]   indexScheme
      restricted:       position.geometry:Grid
      comments: comments
>
<CategoricalRaster
      subclass: CampoGeog_6::MySchema
      attributes:attributeH     Integer(2)
                 attribute7     String(50)
      restricted:position.geometry:CellGrid2D
      comments: comments

>
```

Um *campo geográfico* com representação espacial do tipo *isolinha* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *Field* associada à classe *isolines* do *SAIF*.



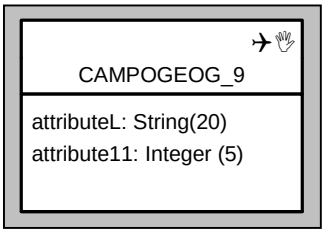

```
<Geographic Object
      subclass: Coverage
      comments: comments
>
<Coverage
   subclass:    Field
   comments:    comments
>
<Field
   subclass:    CampoGeogr_7::MySchema
   attributes:  attributeI      String(15)
                attribute8      Integer(2)
                attribute9      Integer(10)
   restricted:  position.geometry: isolines
   comments:    comments

>
```

Um *campo geográfico* com representação espacial do tipo *grade de pontos* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *Field* associada à classe *pointGrid* do *SAIF*.



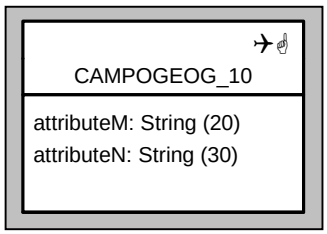

```
<Geographic Object
      subclass: Coverage
      comments: comments
>
<Coverage
      subclass: Field
      comments: comments

>
<Field
    subclass:   CampoGeog_8::MySchema
    attributes: attributeJ  Boolean
                attribute10 Integer(3)
    restricted: position.geometry:PointGrid
    comments:   comments

>
```

Um *campo geográfico* com representação espacial do tipo *TIN* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *Field* associada à classe *TIN* do *SAIF*.




```
<Geographic Object
      subclass: Coverage
      comments: comments
>
<Coverage
      subclass: Field
      comments: comments
>
<Field
      subclass: CampoGeog_9::MySchema
      attributes:attributeL String(20)
                 attribute11 Integer(5)
      restricted:position.geometry:TIN
      comments: comments

>
```

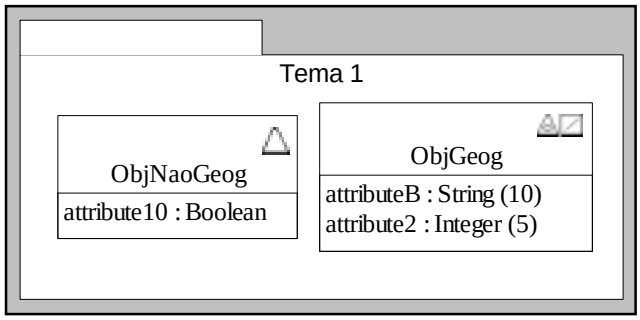
Um *campo geográfico* com representação espacial do tipo *pontos irregulares* no GeoFrame, torna-se na linguagem *SAIFTalk*, uma subclasse de *IrregularDEM* associada a classe *DEMpoint* do *SAIF*.




```
<Geographic Object
      subclass: Coverage
      comments: comments

>

<Coverage
      subclass: Field
      comments: comments

>
```

```
<Field
  subclass:     IrregularDEM
  restricted:   position.geometry.masspoints{}:DEMpoint
  comments:     comments
>
<IrregularDEM
      subclass: CampGeog_10::MySchema
      attributes:attributeM     String(20)
                 attributeN     String(30)
      restricted:position.geometry.masspoints{}:DEMpoint
      comments: comments

>
```

O conceito de *Tema* empregado nos esquemas conceituais, baseados no *framework* GeoFrame, torna-se na linguagem *SAIFTalk*, uma agregação de *GeographicObject* representada explicitamente pela classe *SpatialDataSet*, através cláusula *attributes*, a qual têm a função de agrupar todos os componentes do tema, conforme descrito no *script* a seguir.




```
<GeographicObject
   subclass:    SpatialDataSet
   comments:    comments
>
<SaptialDataSet
   subclass:    Tema 1: MySchema
   attributes:  [geoCompoments] Set(ObjeNaoGeog, ObjGeog)
   comments:    “esse pacote agrupa classes relacionadas em uma
                aplicação”.
>
<GeographicObject
 subclass:   ObjNaoGeog::MySchema
 attibutes:  ID_ObjNaoGeog Integer
             attribute10   Boolean
 restricted: position.geometry:null
 commnets:   comments
>
```

```
<GeographicObject
  subclass: ObjGeog::MySchema
  attributes: ID_ObjGeog        Integer
              attributeB        String(10)
              attribute2        Integer(5)
  restricted: position.geometry:arc
  comments:   comments
>
```

Como definido nesta Seção, a estrutura do esquema genérico de mapeamento pode ser representada de formas distintas. A primeira visa apresentar um fluxograma mais amplo, sem os detalhes de especificação utilizados pela linguagem *SAIFTalk*. A segunda forma de especificação busca detalhar as classes do modelo *SAIF* que são necessárias para contemplar os conceitos definidos do diagrama de classe do GeoFrame.

Conforme apresentado na Seção 2.2.1, o *framework* conceitual GeoFrame adota dez classes diferentes para a representação de fenômenos georreferenciados, sendo seis direcionadas para a representação de campos e quatro para a representação de objetos; além dos conceitos de objeto não geográfico e tema, sendo que cada uma dessas diferentes classes, encontra uma alternativa de codificação adequada na linguagem formal *SAIFTalk*, conforme descrito nessa Seção.

Embora a definição dessas classes esteja no nível conceitual, a escolha de uma delas para representar algum elemento do mundo real indica um caminho para o esquema lógico em uma ferramenta de SIG. Não é por acaso que algumas dessas classes têm correspondentes implementados em SIG comerciais. Embora nem todo SIG possua, simultaneamente, todas as classes de representação definidas tanto pelo *framework* GeoFrame quanto pela linguagem formal adotada.

3.1.2 Regras de Transformação

O segundo grupo de regras refere-se à transformação do esquema conceitual, gerado a partir do *framework* GeoFrame, utilizando a notação gráfica do diagrama de classes da linguagem UML, para os conceitos dos softwares de SIG, disponíveis na biblioteca interna do *FME*.

Neste aspecto, como o modelo de dados dos softwares de SIG (atendidos pelo *FME*) são puramente relacionais, buscou-se adaptar as regras que são definidas por Rumbaugh [RUM 91], que abordam o mapeamento entre modelos O.O e relacionais.

As regras de transformação são especificadas nesta Seção de duas formas distintas. A primeira utiliza o fluxograma ilustrado na Figura 3.2, o qual representa um esquema genérico de aplicação das regras. A segunda forma, utiliza exemplos de diagramas de classes que são traduzidos para a linguagem *SAIFTalk*.

Na Figura 3.2, verifica-se que as classes de um modelo O.O. se tornam tabelas no modelo relacional. Nos casos das associações, dependendo do tipo, são criadas novas tabelas ou as chaves de uma tabela são transportadas para outra tabela. As agregações são transformadas em associações 1:N nos modelos relacionais. Entretanto, o mapeamento da hierarquia de generalização/especialização é mais complexo, devido a existência de representação espacial das classes envolvidas no diagrama, sendo cada

caso descrito em uma determinada regra. A seguir, cada um dos blocos da Figura 3.2 são explorados em detalhe.

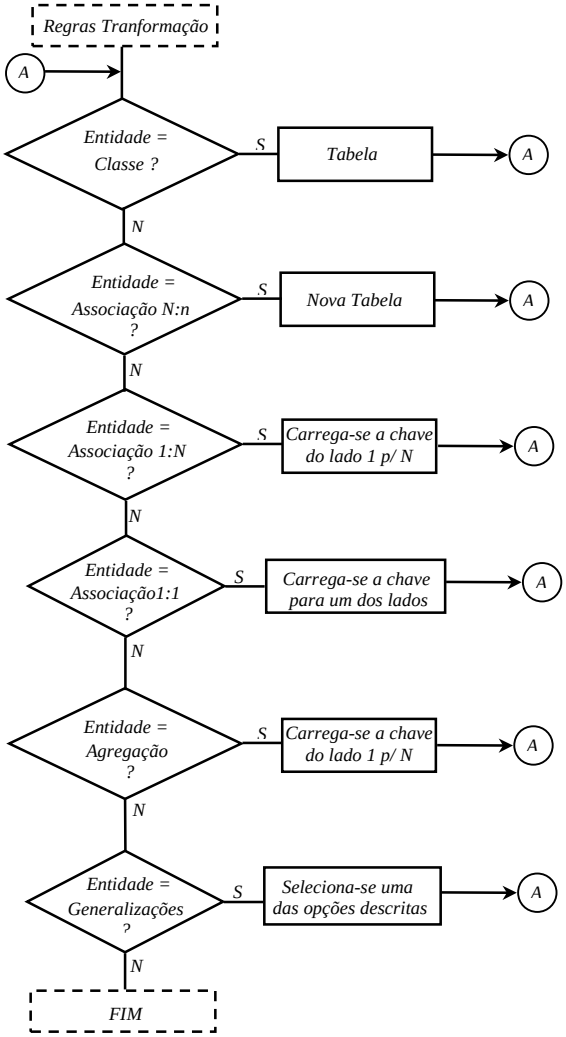
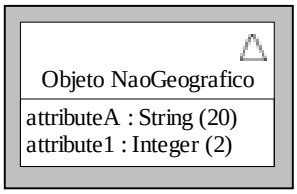


FIGURA 3.2 – Esquema genérico para representar as regras de transformação

3.1.2.1 MAPEAMENTO DE CLASSES E ATRIBUTOS

Segundo Rumbaugh [RUM 91], cada classe, em geral, dá origem a uma tabela, onde cada atributo da classe corresponde a uma coluna, e os valores dos atributos para cada objeto na classe correspondem à dimensão do atributo.

No entanto, deve-se criar uma coluna de identificação, ou seja, uma chave primária para cada tupla armazenada. Em um programa C++, quando ocorre a criação de um novo objeto, é gerado automaticamente o identificador (Id) deste objeto. Já em um banco de dados relacionais, cabe ao projetista a responsabilidade de definir um ou mais atributos como chave primária da tabela. Quando ocorre o mapeamento, recomenda-se armazenar no banco de dados relacional o Id do objeto como chave primária ou qualquer outro atributo do objeto que possa identificá-lo de forma única. A forma de aplicação dessa regra no esquema, implica na criação de um Id para cada classe da aplicação, conforme ilustra o exemplo abaixo:




```
<Geographic Object
  subclass:     ObjetoNaoGeografico::MySchema
  attributtes:  ID_ObjetoNaoGeografico Integer
                attributeA             String(20)
                attribute1             Integer(2)
  restricted:   position.geometry:null
  comments:     comments
>
```

3.1.2.2 MAPEAMENTO DE ASSOCIAÇÃO MUITOS PARA MUITOS

Cada associação “muitos para muitos”, entre objetos, deve ser mapeada em uma nova tabela. A combinação das chaves primárias de cada tabela transforma-se na chave da nova. Essa regra de associação é aplicada neste trabalho, através da inserção de uma nova tabela no esquema, que é formada com as chaves de cada tabela da relação. A Figura 3.3 ilustra um diagrama com essa associação e a seguir esse diagrama é descrito em linguagem *SAIFTalk*.

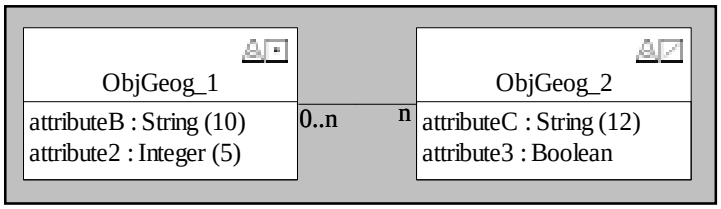


FIGURA 3.3 – Diagrama de classes com associação N:n

```
<Geographic Object
  subclass:     ObjGeog_1::MySchema
  attributtes:  ID_ObjGeog_1 Integer
                attributeB   String(10)
                attribute 2  Integer(5)
  restricted:   position.geometry:point
  comments:     “no caso das associações N:n é  criada automaticamente
                uma nova tabela”. >
<Geographic Object
  subclass:     ObjGeog_2::MySchema
  attributtes:  ID_ObjGeo_2  Integer
                attributeC   String(12)
                attribute3   Boolean
  restricted:   position.geometry:arc
  comments:     “no caso das associações N:n é  criada automaticamente
                uma nova tabela”. >
<Geographic Object
  subclass:     NovaTabela::MySchema
  attributtes:  ID_ObjGeog_1  Integer
                ID_ObjGeog_2  Integer
  comments:     “no caso das associações N:n é  criada
                automaticamente essa tabela”.

 >
```

3.1.2.3 MAPEAMENTO DE ASSOCIAÇÕES UM PARA MUITOS

Nas associações “um para muitos”, existem duas maneiras de realizar o mapeamento para o modelo relacional. No primeiro caso, transpõe-se a chave primária da tabela correspondente à classe do lado ´1´, para a tabela do lado ´muitos´, a partir da criação de uma coluna com a chave da tabela correspondente a uma determinada classe em outra

tabela. No segundo caso, cria-se uma tabela correspondente à associação, transpondo para esta as chaves das duas classes, tornando-se uma associação ´n:N´. As duas formas de mapeamento para o modelo relacional apresentam vantagens.

No primeiro caso, observa-se que uma tabela a menos é apresentada. Isso significa menor espaço ocupado pelos arquivos resultantes e maior eficiência (rapidez) no processamento de operações que envolvam a associação. O segundo caso apresenta uma solução com maior independência de dados, onde cada tabela contém seus dados próprios, considerando que uma mudança estrutural do tipo da associação, passando de ´1:n´ para ´n:N´, não requer nenhuma alteração estrutural nas tabelas. De acordo com as características apresentadas em cada forma de mapeamento, utiliza-se nesse trabalho o primeiro caso, conforme ilustra a Figura 3.4, que envolve a redução de uma tabela, o que torna a aplicação mais rápida no processo de operações. O segundo caso não acrescenta nenhuma diferença em relação ao protótipo desenvolvido neste trabalho, por tratar-se do próprio relacionamento n:N.

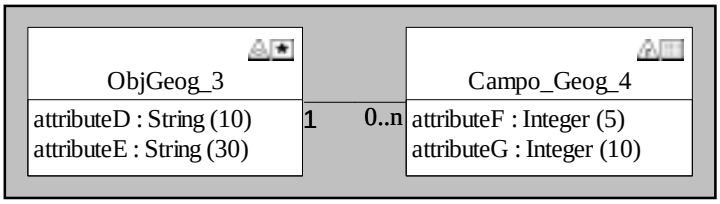


FIGURA 3.4 – Diagrama de classes com associações 1:n

```
<Geographic Object
  subclass:     ObjGeog_3::MySchema
  attributtes:  ID_ObjGeog_3    Integer
                attributeD      String(10)
                attributeE      String(30)
  restricted:   position.geometry:geometricAggregate
  comments:     comments
 >
<Geographic Object
      subclass: Coverage
      comments: comments

>

<Coverage
      subclass: Field
      comments: comments
>
<Field
    subclass:   CampoGeog_4::MySchema
    attributes: ID_ObjGeog_3    Integer
                ID_ObjGeog_4    Integer
                attributeF      Integer(5)
                attributeG      Integer(10)
    restricted: position.geometry:PointGrid
    comments:   comments

>
```

3.1.2.4 MAPEAMENTO DE ASSOCIAÇÕES UM PARA UM

Nas associações “um para um” deve-se criar uma tabela para cada classe e transpor a chave de uma tabela para a outra. O sentido da transposição depende das formas de

acesso para o ganho de eficiência. Essa regra de mapeamento é aplicada neste trabalho, através da decisão do projetista em definir quais as tabelas responsáveis por enviar e receber a chave primária. No exemplo da Figura 3.5 , o objeto identificador da classe *objeto não geográfico* é transposto para a classe *campo_Geog_5*.

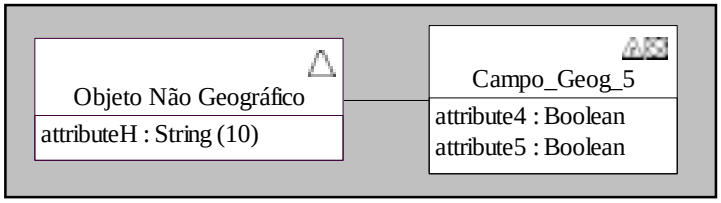


FIGURA 3.5 – Diagrama de classes com associações 1:1

```
<Geographic Object
  subclass:     ObjetoNaoGeografico::MySchema
  attributtes:  ID_ObjetoNaoGeografico  Integer
                attributeH              String(10)
  restricted:   position.geometry:null
  comments:     comments
 >
<Geographic Object
      subclass: Coverage
      comments: comments

>

<Coverage
      subclass: Field
      comments: comments
>
<Field
      subclass: Campo_Geog_5::MySchema
      attributes:ID_ObjetoNaoGeografico Integer
                ID_Campo_Geog_5         Integer
                attribute4              Boolean
                attribute5              Boolean
      restricted:position.geometry:TIN
      comments: “o ID transposto deve ser sempre o primeiro da lista
                        dos atributos”.

>
```

3.1.2.5 MAPEAMENTO DE GENERALIZAÇÃO/ESPECIALIZAÇÃO

No *framework* conceitual GeoFrame, as abstrações de generalização e especialização se aplicam tanto a classes georeferenciadas quanto a classes convencionais, seguindo as definições propostas pelo padrão UML, em que o triângulo conecta a superclasse a suas subclasses. Em [RUM91] são definidas três regras básicas para o mapear essas generalizações/especializações em tabelas:

Regra 1 - a classe generalizada e as classes especializadas são mapeadas individualmente para cada tabela. O *Id* da classe generalizada é reproduzido nas tabelas das classes especializadas, sendo a identidade do objeto preservada através do *Id* compartilhado. Um novo atributo *Type* deve ser criado na superclasse, com a finalidade de possibilitar determinadas consultas.

Regra 2 - cada classe especializada é mapeada para uma tabela. A classe de generalização é eliminada e seus atributos são herdados em cada tabela de especialização;

Regra 3 - a classe generalizada é mapeada para uma tabela, juntamente com os atributos das classes especializadas, sendo necessário a criação de um novo atributo *Type* para facilitar as consultas.

Qualquer uma das regras, descritas anteriormente, podem ser utilizadas com algumas vantagens e desvantagem em aplicações convencionais. Entretanto, quando se trata de aplicações geográficas, torna-se necessário verificar o tipo de representação de cada classe, através dos esteriótipos do GeoFrame.

Para selecionar a regra de generalização/especialização que melhor se adapte às necessidades da aplicação, deve-se considerar as variações quanto a localização dos esteriótipos, os quais podem ser modelados tanto na superclasse como nas subclasses. A consequência da variação de localização dos esteriótipos pode estar diretamente relacionada à modelagem do projetista ou à abordagem da aplicação. A Figura 3.6 resume os seis tipos de situações que podem ocorrer na modelagem de esquemas conceituais em função das regras generalização/especialização e define quais regras de podem ser empregadas em cada caso.

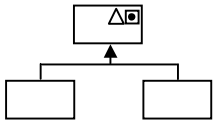
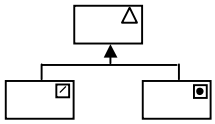
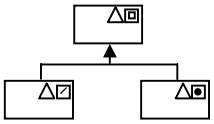
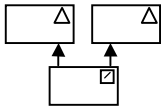
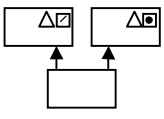
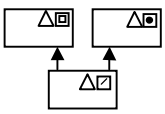
Situações		*Regra 1*	*Regra 2*	*Regra 3*
1		X	X	X
2		X	X	
3		X		
4		X	X	X
5		X		X
6		X		

FIGURA 3.6 – Exemplos genéricos de aplicação de regras generalização/especialização

Conforme ilustra a Figura 3.6, as classes envolvidas no processo generalização/especialização com a existência de esteriótipos são encontradas facilmente nos esquemas conceituais, baseados no GeoFrame. Todas as regras definidas

em [RUM 91] são empregadas nesse trabalho, entretanto, apenas a *situação 1* da Figura 3.6 está especificado no protótipo. As situações abordadas na Figura 3.6 são especificadas através de exemplos genéricos:

- *Situação 1 - subclasses sem esteriótipos*

Esse exemplo trata de modelagem de aplicações geográficas, onde a representação espacial dos fenômenos geográficos ou objetos não geográficos são definidos diretamente na superclasse. Nesse caso, é possível empregar qualquer uma das regras de generalização/especialização, conforme definido em [RUM 91]. Porém, na utilização das regras (1) e (3) deve-se manter a representação espacial inserida na respectiva classe, conforme descrito no *script* 2.

Entretanto, de acordo com a estrutura da regra (2), as subclasses herdam o *Id*, atributos e métodos da superclasse, adaptando no caso de aplicações geográficas, o conceito de herança para a representação espacial dos fenômenos geográficos e para os objetos não geográficos, conforme ilustra os diagramas de classes da Figura 3.7, os quais são baseados no *framework* conceitual GeoFrame.

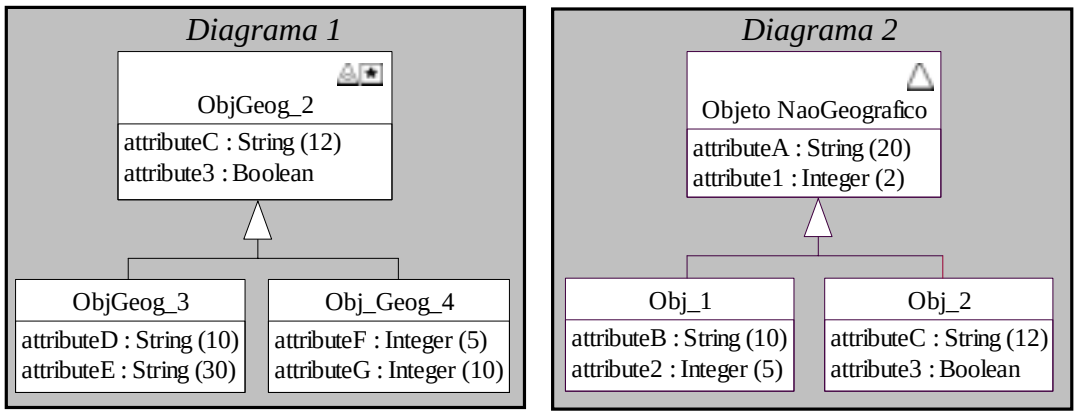


FIGURA 3.7 – Diagramas de classes com esteriótipos nas superclasses

Os diagramas 1 e 2 ilustrados na Figura 3.7 são descritos nos *scripts* 1 e 2. O diagrama 1 tem como base as características apresentadas na regra 1 e considera a permanência da representação espacial dos objetos na superclasse. O diagrama 2 é baseado na definição da regra 2 e considera o conceito de herança utilizado na representação espacial dos objetos.

Script 1: aplicação da regra 1 no diagrama 1 da Figura 3.7

```
<Geographic Object
  subclass:     ObjGeog_2::MySchema
  attributtes:  ID_ObjGeog_2            Integer
                attributeC              String(12)
                attribute3              Boolean
                type                    String
  restricted:   position.geometry:geometricAggregate
  comments:     "as subclasses compartilham o Id da superclasse, mas
                cada classe implementa os seus próprios atributos e a
                representação espacial é mantida na superclasse. O
                atributo type é criado para facilitar as consultas".
 >
<ObjGeog_2::MySchema
```

```
  subclass:     ObjGeog_3::MySchema
  attributtes:  ID_ObjGeog_2            Integer
                attributeD              String(10)
                attributeE              String(30)
  restricted:   position.geometry:null
  comments:     comments
>
<ObjGeog_2::MySchema
  subclass:     ObjGeog_4::MySchema
  attributtes:  ID_ObjGeog_2            Integer
                attributeF              Integer(5)
                attributeG              Integer(10)
  restricted:   position.geometry:null
  comments:     comments
>
```

Script 2: aplicação da regra 2 no diagrama 2 da Figura 3.7

```
<Geographic Object
  subclass:     Obj_1::MySchema
  attributtes:  ID_ObjetoNaoGeografico  Integer
                attributeA              String(20)
                attribute1              Integer(2)
                attributeB              String(10)
                attribute2              Integer(5)
  restricted:   position.geometry:null
  comments:     "as subclasses herdam o Id, os atributos e
                representação espacial da superclasse".
>
<Geographic Object
  subclass:     Obj_2::MySchema
  attributtes:  ID_ObjetoNaoGeografico  Integer
                attributeA              String(20)
                attribute1              Integer(2)
                attributeC              String(12)
                attribute3              Boolean
  restricted:   position.geometry:null
  comments:     "as subclasses herdam o Id, os atributos e
                representação espacial da superclasse".
>
```

- *Situação 2 - subclasses com tipos diferentes de esteriótipos*

Outra situação de modelagem de aplicações geográficas trata da existência de diferentes tipos de representações espaciais nas subclasses, para esse caso, é possível empregar as regras (1) e (2). A regra (3) não é empregada devido a impossibilidade de mapear a existência de múltiplas representações para a linguagem *SAIFTalk*.

A Figura 3.8 ilustra a existência de diferentes representações espaciais para cada subclasse e os *scripts* 3 e 4 demonstram a aplicação das regras (1) e (2), respectivamente.

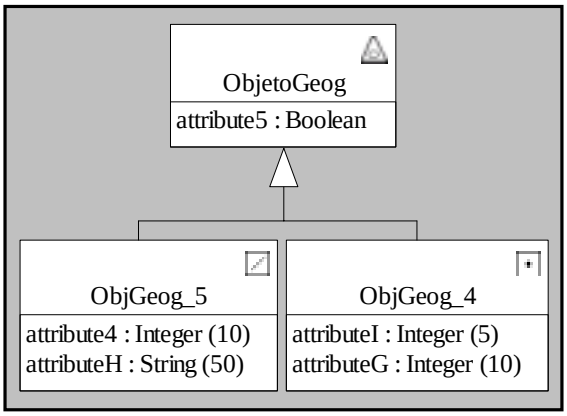


FIGURA 3.8 – Diagrama de classes com diferentes esteriótipos nas subclasses

Script 3: aplicação da regra 1 na Figura 3.8

```
<Geographic Object
  subclass:     ObjetoGeog::MySchema
  attributtes:  ID_ObjetoGeog   Integer
                attribute5      Boolean
                type            String
  restricted:   position.geometry:null
  comments:     “o atributo TYPE distingue as subclasses”.
 >
 <ObjetoGeog::MySchema
  subclass:     ObjGeog_5::MySchema
  attributtes:  ID_ObjetoGeog           Integer
                attribute4              Integer(10)
                attributeH              String(50)
  restricted:   position.geometry:arc
  comments:     commnets
>
<ObjetoGeog::MySchema
 subclass:      ObjGeog_4::MySchema
  attributtes:  ID_ObjetoGeog           Integer
                attributeI              Integer(5)
                attributeG              Integer(10)
  restricted:   position.geometry:point
  comments:     commnets

>
```

Script 4: aplicação da regra 2 na Figura 3.8

```
<Geographic Object
  subclass:     ObjGeog_5::MySchema
  attributtes:  ID_ObjetoGeog   Integer
                attribute5      Boolean
                attribute4      Integer(10)
                attributeH      String(50)
  restricted:   position.geometry:arc
  comments:     “ a superclasse é mapeada para cada subclasse,
                juntamente com o seu atributo e o ID”.
 >
<Geographic Object
  subclass:     ObjGeog_4::MySchema
  attributtes:  ID_ObjetoGeog   Integer
                attribute5      Boolean
```

```
                attributeI      Integer(5)
                attributeG      Integer(10)
  restricted:   position.geometry:point
  comments:     “a superclasse é mapeada para cada subclasse,
                juntamente com o seu atributo e o ID”
.
>
```

- *Situação 3 - subclasses e superclasse com diferentes tipos de esteriótipos*

A Figura 3.9 ilustra um diagrama de classes genérico, com diferentes representações para cada classe do esquema. Nessa situação é possível empregar a regra (1). Porém, não é possível utilizar às regras (2) e (3), devido a ocorrência de múltiplas representações para uma mesma classe. Portanto, o *script* abaixo descreve a utilização da regra 1 para a Figura 3.9

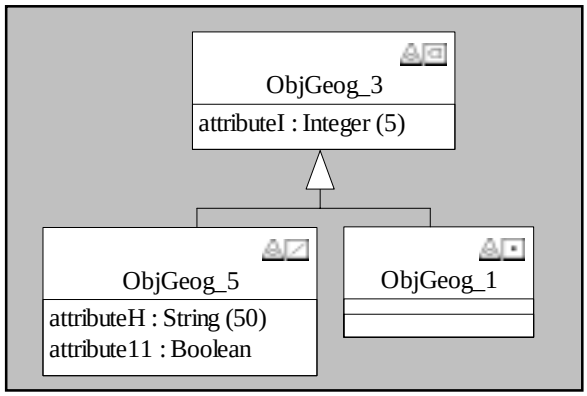


FIGURA 3.9 – Diagrama de classes com diferentes esteriótipos nas subclasses e superclasse

Script 5: aplicação da regra 1 na Figura 3.9

```
<Geographic Object
  subclass:     ObjGeog_3::MySchema
  attributtes:  ID_ObjGeog_3    Integer
                attributeI      Integer(5)
                type            String
  restricted:   position.geometry:polygon
  comments:     “cada classe implementa o sua representação espacial”.
 >
 < ObjGeog_3::MySchema
  subclass:     ObjGeog_5::MySchema
  attributtes:  ID_ObjGeog_3            Integer
                attributeH              String(50)
                attribute11             Boolean
  restricted:   position.geometry:arc
  comments:     “cada classe implementa o sua representação espacial”
>
< ObjGeog_3::MySchema
  subclass:     ObjGeog_1::MySchema
restricted:     position.geometry:point
  comments:     “cada classe implementa o sua representação espacial”
>
```

- *Situação 4 - herança múltipla com esteriótipo na subclasse*

Outro tipo de conceito que pode estar presente na modelagem baseada no *framework* conceitual GeoFrame e é existente no padrão *SAIF*, trata-se da aplicação do conceito de herança múltipla. Nesse caso, são demonstrados três possíveis situações que podem ocorrer em um esquema conceitual, baseado no GeoFrame, e as respectivas regras que podem ser empregadas em cada situação.

Na situação ilustrada, na Figura 3.10, é possível empregar qualquer uma das regras descritas em [RUM 91] desde que a representação espacial se mantenha nas respectivas classes, quando aplicada as regras (1) ou (2). A regra (3) é demonstrada através do *script* abaixo:

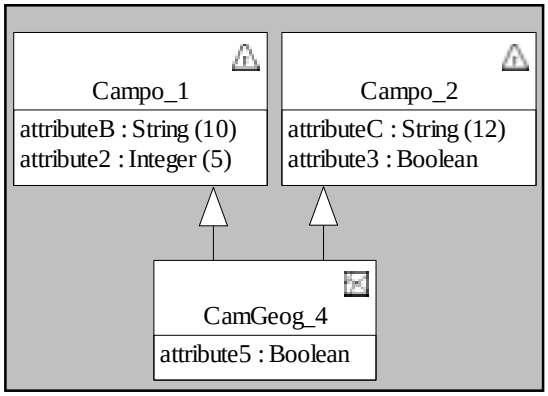


FIGURA 3.10 – Diagramas de classes com herança múltipla e esteriótipo na subclasse

Script 6: aplicação da regra 3 na Figura 3.10

```
<Geographic Object
      subclass:  Coverage
      comments:  comments
>
<Coverage
      subclass:  Raster
      comments:  comments
>
<Raster
      subclass:          CategoricalRaster
      attributes:        [indexScheme]   indexScheme
      restricted:        position.geometry:Grid
      comments:          comments

>

<CategoricalRaster
      subclass:          Campo_1::MySchema
      attributes:        attribute5      Boolean
                         ID_Campo_1      Integer
                         attributeB      String(10)
                         attribute2      Integer(5)
                         type            String
      restricted:        position.geometry:CellGrid2D
      comments:          " as características da subclasse migram para
                         as superclasse".
>
<Geographic Object
      subclass:  Coverage
```

```
      comments:  comments
>
<Coverage
      subclass:  Raster
      comments:  comments
>
<Raster
      subclass:          CategoricalRaster
      attributes:        [indexScheme]   indexScheme
      restricted:        position.geometry:Grid
      comments:          comments
>
<CategoricalRaster
      subclass:          Campo_2::MySchema
      attributes:        attribute5      Boolean
                         ID_Campo_2      Integer
                         attributeC      String(12)
                         attribute3      Boolean
      restricted:        position.geometry:CellGrid2D
      comments:          comments
>
```

- *Situação 5 - herança múltipla com esteriótipos nas superclasses*

No caso da Figura 3.11 pode-se empregar as regras (1) e (3). A regra (2) não é permitida devido a impossibilidade de mapear a existência de múltiplas representações para uma mesma classe em linguagem *SAIFTalk*. O *script* abaixo descreve a utilização da regra (3), onde as representações espaciais devem ser mantidas nas superclasses.

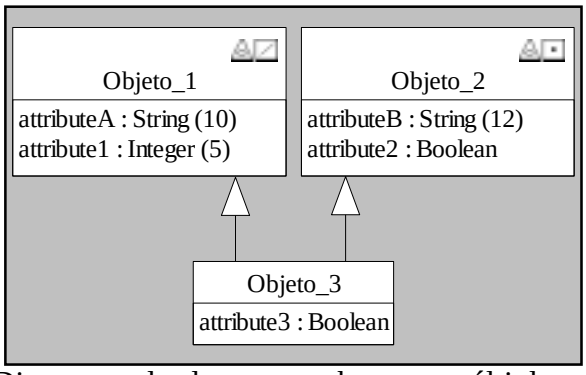


FIGURA 3.11 – Diagramas de classes com herança múltipla com esteriótipos nas superclasses

Script 7: aplicação da regra 3 na Figura 3.11

```
<Geographic Object
  subclass:     Objeto_1::MySchema
  attributtes:  attribute3      Boolean
                ID_Objeto_1     Integer
                attributeA      String(10)
                attribute1      Integer(5)
                type            String
  restricted:   position.geometry:arc
  comments:     comments
>
 <Geographic Object
  subclass:     Objeto_2::MySchema
  attributtes:  attribute3      Boolean
                ID_Objeto_2     Integer
```

```
                attributeB      String(12)
                attribute2      Boolean
  restricted:   position.geometry:point
  comments:     commnets
>
```

➢ *Situação 6 - herança múltipla com diferentes esteriótipos nas superclasses e subclasse*

Outro exemplo de diagrama de classe referente à herança múltipla é ilustrado na Figura 3.12. Esse diagrama apresenta diferentes tipos de representações espaciais para cada classe do esquema, nessa situação é possível utilizar a regra (1), conforme demonstra o *script* a seguir. As regras (2) e (3) não podem ser empregadas, devido a geração de múltiplas representações, como discutido anteriormente.

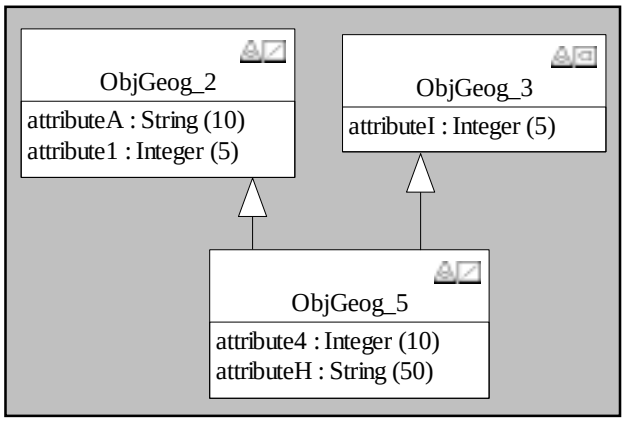


FIGURA 3.12 – Diagrama de classes com herança múltipla e diferentes esteriótipos nas superclasses e subclasses

Script 8: aplicação da regra 1 na Figura 3.12

```
<Geographic Object
  subclass:     ObjGeog_2:MySchema
  attributtes:  ID_ObjGeog_2    Integer
                attributeA      String(10)
                attribute1      Integer (5)
                type            String
  restricted:   position.geometry:arc
  comments:     “o atributo TYPE é utilizado em casos em que a
                superclasse apresenta outras subclasses”.
 >
 <Geographic Object
  subclass:     ObjGeog_3::MySchema
  attributtes:  ID_ObjGeog_3    Integer
                attributeI      Integer (5)
                type            String
  restricted:   position.geometry:polygon
  comments:     “cada classe implementa sua própria representação
                espacial, sendo a identidade do objeto preservada
                através do Id compartilhado”

>
<ObjGeog_2:MySchema, ObjGeog_3::MySchema
  subclass:     ObjGeog_5::MySchema
  attributtes:  ID_ObjGeog_2    Integer
                ID_ObjGeog_3    Integer
```

```
                attribute4      Integer(10)
                attributeH      String(50)
  restricted:   position.geometry:arc
  comments:     “cada classe implementa sua própria representação
                espacial, sendo a identidade do objeto preservada
                através do Id compartilhado”
>
```

Existem algumas vantagens e desvantagens em utilizar as regras de generalização/especialização defindas em [RUM 91], independentemente da localização dos esteriótipos no diagrama.

No caso da regra (1), o mapeamento é considerado mais correto, contudo, envolve muitas tabelas e a navegação das classes de generalização para as classes de especialização pode ser lenta. A regra (2) apresenta uma abordagem que pode ser utilizada se as classes especializadas tiverem muitos atributos e a classe generalizada tiver poucos. A regra (3) é considerada uma abordagem menos satisfatória, porém pode ser útil se existirem somente duas ou três classes de generalização com poucos atributos.

Dessa forma, os dois últimos casos são abordagens alternativas, motivadas pelo desejo de eliminar a navegação da classe de generalização para as classes de especialização, melhorando o desempenho de consultas, contudo, essa melhora de desempenho pode ocasionar outros problemas como, armazenamento de valores nulos.

3.1.2.6 MAPEAMENTO DE AGREGAÇÃO

Uma definição presente nos esquemas conceituais geográficos, baseados no *framework* conceitual GeoFrame, trata do conceito de agregação. Esse conceito, segundo Navathe [NAV 92], pode ser definido de duas formas distintas. Na primeira forma, um objeto é definido como uma agregação dos atributos que o descrevem, ou seja, um objeto é composto de um conjunto de objetos atômicos. Por outro lado, o construtor de agregação permite representar relacionamentos do tipo “é_parte_de”, onde um objeto complexo é definido como uma agregação de suas partes.

Neste trabalho, uma agregação representa a idéia de que uma entidade consiste de outras entidades. Uma agregação pode ocorrer entre classes convencionais, entre classes georreferenciadas ou entre uma classe convencional e uma classe georreferenciada.

Desta forma, uma agregação é transformada diretamente em uma associação ´1:N´, onde a chave do lado ´1´ transpõe-se para o lado `N´. Essa regra está explicitamente definida em [AMB 2000], [BED 98] e [RAT 2000a].

Segundo Davis [DAV 2000] quando a agregação ocorre entre classes georreferenciadas, é necessário empregar o conceito de *agregação espacial*.

Uma agregação espacial indica que a representação espacial dos fenômenos geográficos de cada parte deve estar totalmente contida na geometria do todo, sendo representada de forma diferenciada da convencional. Neste trabalho o conceito de agregação espacial não é abordado, devido a falta de expressão da linguagem *SAIFTalk*.

O diagrama de classes da Figura 3.13, expressa o conceito de agregação utilizado neste trabalho e a sua regra é descrita em linguagem *SAIFTalk*.

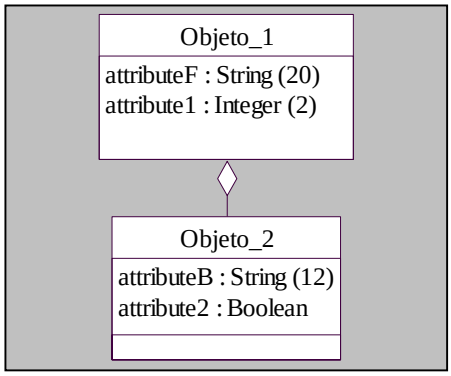


FIGURA 3.13 – Diagrama de classes com agregação

```
<Geographic Object
  subclass:     Objeto_1::MySchema
  attributtes:  ID_Objeto_1     Integer
                attributeF      String(20)
                attribute1      Integer(2)
  restricted:   position.geometry:null
  comments:     “a chave do lado ´1´ é transposta para a classe do
                lado ´n´”
>
 <Geographic Object
  subclass:     Objeto_2::MySchema
        ID_Objeto_1     Integer
  attributtes:  ID_Objeto_2     Integer
                attributeB      String(12)
                attribute2      Boolean
  restricted:   position.geometry:null
  comments:     “a chave do lado 1 é transposta para a classe do lado
                n”.

>
```

Tanto as definições das regras descritas nas Seções 3.3.1 e 3.3.2 são aplicadas de forma integrada para o mapeamento do esquema conceitual baseado no GeoFrame para a linguagem formal do padrão *SAIF*, e desta linguagem para os modelos lógicos dos softwares de SIG através da utilização da ferramenta de tradução apresentada no próximo Capítulo.

4 Protótipo de um Ambiente de Suporte para Aplicação das Regras de Mapeamento

Este Capítulo trata do desenvolvimento do protótipo[4] e visa demonstrar, através da prática, que as regras de mapeamento, definidas no Capítulo 3 são realmente funcionais e necessárias, visto que a tarefa de elaborar um esquema conceitual baseado no diagrama de classe do padrão *SAIF*, torna-se uma tarefa muito complexa, devido principalmente ao elevado grau de especificação do modelo dentre outros aspectos, conforme apresentado no Capítulo 3.

Neste caso, é demonstrada a possibilidade de realizar o mapeamento de esquemas conceituais de SIG, gerados a partir do *framework* conceitual GeoFrame, em esquemas lógicos proprietários dos diversos SIG. Esse objetivo é alcançado através da utilização das regras de mapeamento GeoFrame-*SAIF* e do desenvolvimento de uma arquitetura de suporte, ilustrada na Figura 4.1, a qual é composta pelas ferramentas *RationalRose2000* e *FME2000.*

O protótipo, baseado nessa arquitetura, define um *Gerador de Código SAIF*, que fornece ao projetista recursos automatizados para transformar esquemas conceituais em *scripts SAIF* compatíveis com um conjunto de regras, as quais estão embutidas no programa gerador de código. A garantia de que as regras estão estruturadas corretamente é o arquivo gerado como resultado final de todo processo.

A finalidade deste Capítulo é descrever uma arquitetura de um ambiente de suporte, o funcionamento do protótipo e exemplificá-lo. Para tanto, está dividido em quatro seções. A Seção 4.1 tem por objetivo descrever as ferramentas necessárias para a elaboração de um ambiente de suporte. A Seção 4.2 busca descrever as características gerais do protótipo. A Seção 4.3 refere-se aos passos executados para a implementação do protótipo, desde a fase da elaboração do esquema conceitual até a inserção do modelo de dados no software de SIG selecionado. Para finalizar, na Seção 4.4 é ilustrado um exemplo de aplicação do mapeamento, demonstrando a aplicabilidade do mecanismo.

4.1 Arquitetura de Suporte para Mapeamento

Esta Seção define uma estrutura integrada de ferramentas que são utilizadas para elaborar o protótipo, a fim de automatizar o processo de mapeamento de esquemas conceituais para os esquemas lógicos dos softwares de SIG.

Conforme a Figura 4.1, as regras de mapeamento definidas no Capítulo 3 são inseridas no novo módulo da ferramenta *RationalRose2000* (Seção 2.3.1). Esse módulo

[4] A prototipação de software deve ser desenvolvida rapidamente, de forma que o cliente possa avaliar o protótipo e recomendar mudanças [THO 98].

denominado *Gerador de Código SAIF* permite gerar um *script SAIF* a partir de um esquema conceitual baseado no *framework* conceitual GeoFrame.

Na Figura 4.1 é possível observar que o resultado desse mapeamento no *Rose* é um arquivo *SAIF*, o qual é interpretado pela ferramenta *FME* (Seção 2.3.2), gerando automaticamente um esquema lógico de software SIG, através da utilização do módulo tradutor semântico *SAIF* [FME 2000].

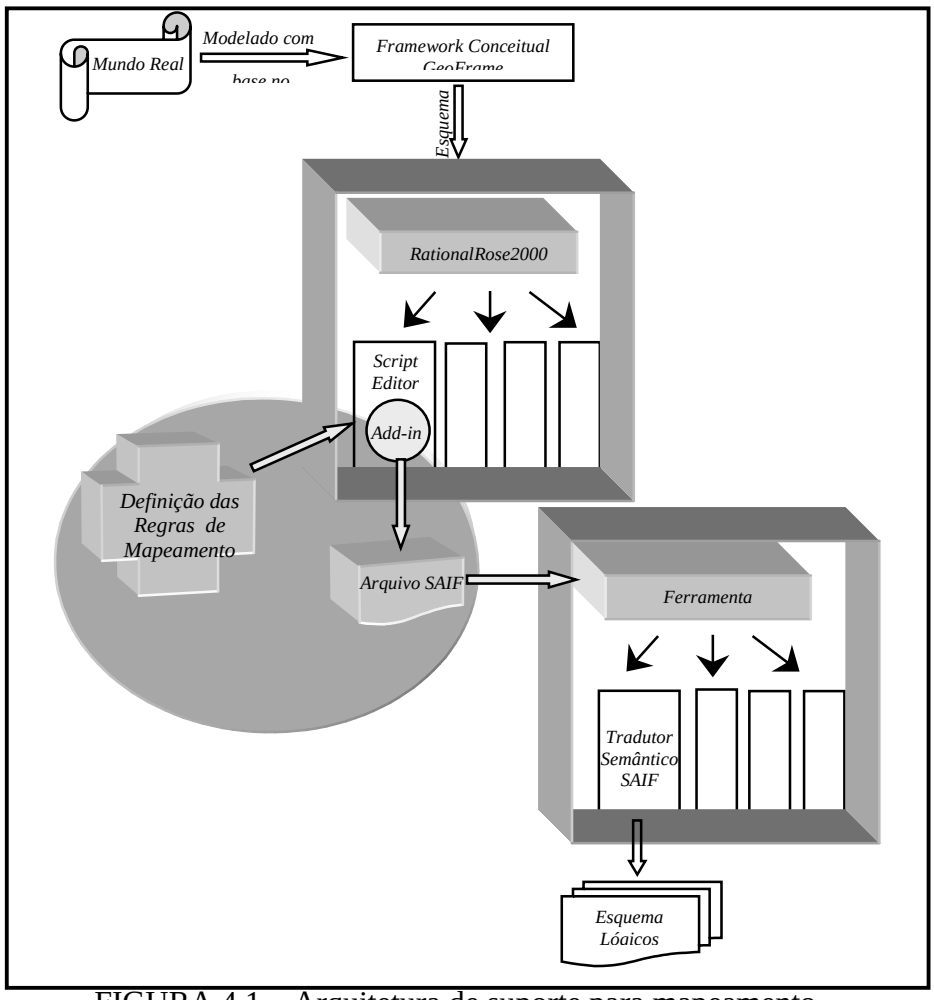


FIGURA 4.1 – Arquitetura de suporte para mapeamento

4.2 Descrição Geral

Esta Seção apresenta o propósito do *Gerador de Código SAIF*, uma descrição informal do módulo, um diagrama de contexto e, por fim, um resumo dos requisitos. Essa fase tem por objetivo, além da identificação e explicitação desses requisitos, a definição do escopo desse Capítulo.

4.2.1 Propósito do Gerador de Código SAIF

O principal propósito do *Gerador* é verificar a funcionalidade das regras de mapeamento, além de aprimorar o *framework* GeoFrame, através da definição de um *script* correspondente aos esquemas conceituais elaborados no *Rose* pelo projetista.

4.2.2 Modelo descritivo do *Gerador*

O *Gerador* é um programa criado através dos recursos da ferramenta *RationalRose2000* que, após compilado, torna-se um módulo pertencente ao próprio *Rose*. A tarefa do projetista se resume em elaborar o esquema conceitual da aplicação, com base no *framework* conceitual GeoFrame e selecionar a opção desejada.

O protótipo fornece a opção de mapear o esquema conceitual para a linguagem do padrão *SAIFTalk* através do submenu *Gerador de Código SAIF* (Anexo 9). Esse submenu ativa o programa, o qual solicita ao projetista alguns dados referentes à aplicação em questão. Esses dados são selecionados ou preenchidos de forma a tornar o *script* final mais ajustado à realidade da aplicação, cabendo ao projetista tomar algumas decisões de como melhor definir a sua aplicação, com base em regras pré-definidas no programa. Desta forma, o projetista aceita ou altera a sugestão oferecida pelo programa ou ainda, seleciona a opção que melhor se encaixa naquela situação em questão.

É disponibilizado ao projetista, dentro desse submenu, um módulo de verificação de consistência capaz de indicar os pontos do diagrama discordantes com a linguagem UML. Este módulo pertence à estrutura interna do próprio *Rose*, sendo disponibilizado ao projetista. Outra característica desse submenu é a descrição da ajuda *on-line*, que procura explicar os passos que o projetista deve seguir para mapear o seu esquema conceitual para o esquema lógico dos software de SIG.

4.2.3 Diagrama de Contexto do Módulo

O diagrama é uma visão de alto nível do módulo, na qual procura-se identificar as situações que implicam decisões do projetista para tornar a sua aplicação mais eficiente. Essas decisões tornam o módulo mais flexível, pois o projetista influencia diretamente no modelo final da aplicação. O diagrama de contexto baseia-se em perguntas que são respondidas pelo projetista, através de telas, durante a geração do arquivo, sendo que, as respostas obtidas são inseridas no programa, gerando um *script SAIF* como o resultado de todo o processo. A Figura 4.2 ilustra esse diagrama.

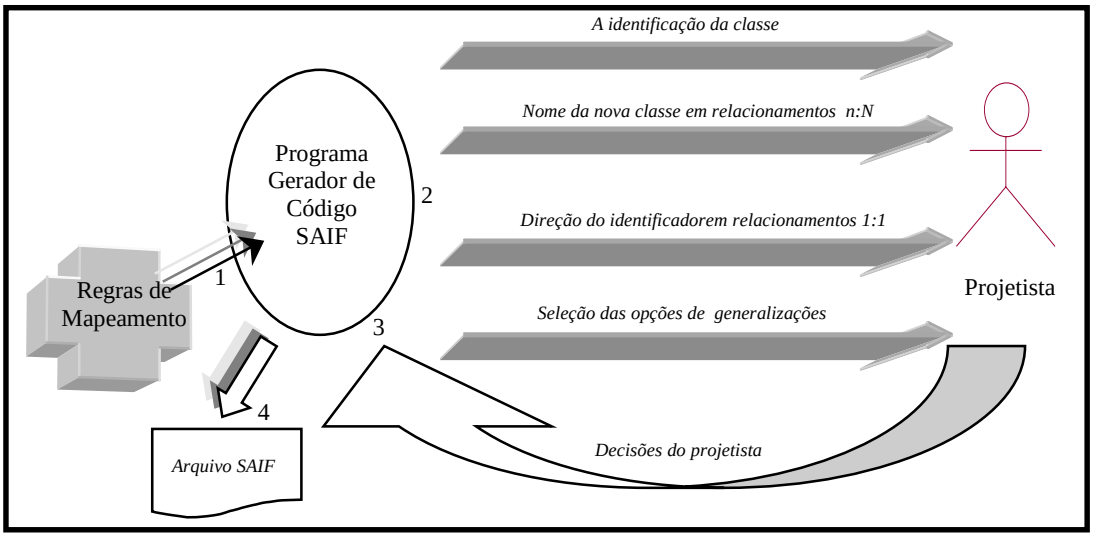


FIGURA 4.2 – Diagrama de contexto do módulo

4.2.4 Resumo dos Requisitos

Esta Seção descreve um resumo dos requisitos que o módulo atende. Esses itens são decorrentes da análise do modelo descritivo do módulo e do diagrama de contexto. Os principais tópicos que esse módulo atende são:

- encapsular as regras de correspondência definidas na Seção 3.1.1;
- encapsular as regras de transformação apresentadas na Seção 3.1.2;
- fornecer um mecanismo para conversão automática de esquemas conceituais baseados no GeoFrame para a linguagem *SAIFTtalk*;
- fornecer um mecanismo para gerar ajuda *on-line*;
- fornecer um arquivo no formato *SAIF*.

4.3 Especificação Detalhada

A descrição formal envolve todo o processo de elaboração do protótipo, desde sua fase inicial, onde os elementos do mundo real são abstraídos para o esquema conceitual, baseado no *framework* GeoFrame, até a definição do modelo de dados no software selecionado pelo projetista.

4.3.1 Elaboração do Esquema Conceitual

A construção de um esquema conceitual é parte fundamental de qualquer processo de desenvolvimento de sistemas de informação, seja geográfico ou convencional. Além disso, o sucesso da implementação de um sistema é dependente da qualidade da modelagem dos objetos do mundo real e suas interações com o banco de dados.

Durante o processo de modelagem de esquemas conceituais geográficos é necessário identificar todos os objetos do mundo real que, de alguma forma, interfiram no sistema que se está procurando modelar. Em seguida é preciso considerar, não apenas o conjunto de dados descritivos de um objeto e seus relacionamentos com os demais, mas também deve-se considerar a escolha de uma representação espacial a adotar para cada objeto geográfico, determinando as características essenciais para que seu comportamento ou funcionamento sejam adequadamente incorporados ao sistema.

Os dados geográficos possuem características peculiares que fazem com que sua modelagem seja mais complexa do que a de dados convencionais. Modelar os aspectos espaciais é de fundamental importância na criação de um banco de dados geográficos, principalmente porque se está modelando a realidade geográfica, onde a visão que os usuários têm do mundo real pode variar, dependendo do que eles necessitam representar e do que eles esperam extrair dessa representação. Nesse contexto, conceitos como a geometria tornam-se decisivos para o processo de entrada de dados em um software de SIG [DAV 99].

A incorporação da geometria nos fenômenos espaciais consiste em escolher uma representação adequada para cada um deles, que seja capaz de incorporar suas características espaciais como localização e forma geométrica. No *framework* conceitual

GeoFrame existem duas grandes classes que generalizam essas formas de representação geométrica: a classe *representaçãoCampo*, adequada para representar fenômenos de variação contínua no espaço, e a classe *objetoEspacial*, adequada para representar entidades individualizáveis, ou seja, que possuem identidade própria e suas características podem ser descritas através de atributos [LIS 2000].

O componente espacial da *representaçãoCampo* pode ser representado em um sistema de informação geográfica de diversas maneiras como, por exemplo, através de modelos digitais de terreno (DTM – *Digital Terrain Models*), matrizes de pixels e outros. O componente espacial de um *objetoEspacial* é, usualmente, representado por meio de uma forma geométrica simples como, ponto, linha ou polígono.

O projetista deve construir o esquema conceitual de sua aplicação de SIG conforme o método definido pelo *framework* conceitual GeoFrame em [LIS 2000], bem como a incorporação das representações espaciais definidas por ele. Entretanto, existem algumas limitações na tradução do esquema conceitual para o modelo lógico implementados pelos SIG, que se referem ao ambiente de suporte proposto.

Uma das limitações é relativa às múltiplas representações espaciais para uma mesma classe. Essa situação deve ser evitada durante o processo de modelagem de esquemas conceituais, devido à falta de especificação de uma cláusula pela linguagem *SAIFTalk* que comporte essa situação. Dessa forma, aconselha-se ao projetista definir a escala ou projeção na qual o esquema deve ser implementado, decidindo-se por apenas um tipo de representação para cada fenômeno geográfico da aplicação.

Outra limitação surge quando insere-se um arquivo na ferramenta *FME*, contendo um *script SAIF* correspondente ao conceito de *tema* do GeoFrame. Esse *script* não é identificado pelo *FME* que interrompe o processo de tradução automática. Embora, o conceito de *tema* apresentado pelo GeoFrame possa ser modelado, conforme descrito em [LIS 2000], e apresente correspondência tanto no padrão *SAIF* como nos modelos lógicos dos softwares de SIG, é colocado à margem durante o processo de mapeamento para a linguagem *SAIFTalk*, devido a falta de reconhecimento pelo *FME*.

Ao término da modelagem, o projetista deve certificar-se que estão definidos corretamente: as cardinalidades das associações, os tipos de domínios e os valores de seus atributos. Esses itens devem ser verificados antes da execução do *submenu Geração de Código SAIF*, com a finalidade de garantir a eficiência na geração do arquivo.

4.3.2 Geração de Código *SAIF*

Como referido anteriormente, para a concretização do mapeamento de esquemas conceituais para os esquemas lógicos que os softwares de SIG implementam, torna-se necessária, primeiramente, a definição de um *script* para os esquemas conceituais, os quais são gerados a partir do *framework* GeoFrame.

A escolha para a definição desse *script* baseia-se na ferramenta *FME* (Seção 4.2), que é capaz de gerar, automaticamente, a partir de um *script*, os modelos de dados proprietários de diversos SIG.

Para gerar esse *script* com base no padrão *SAIF*, denominado *SAIFTalk*, faz-se necessário que o esquema conceitual elaborado pelo projetista esteja disponível na ferramenta *RationalRose2000*. Essa ferramenta permite elaborar um *Add-in* para geração de código em qualquer formato. Nessa pesquisa, o formato selecionado é o *SAIF*, devido às características apresentadas acima.

Um *Add-in* permite a configuração e a automatização de muitas das características do *Rose*. Esse componente é uma coleção de algumas combinações das seguintes características: funcionalidades; itens do menu principal; esteriótipos; ajuda *on-line*; itens de menu rápido; propriedades; tipos de dados; ajuda ao contexto; configuração de especificações e tratamento de eventos.

Das características citadas acima, este trabalho abrange as quatro primeiras, pois as demais, relacionam-se diretamente ao módulo de definição dos diagramas de classes, não interferindo e nem colaborando na geração de código. A pesquisa não tem por objetivo a exploração de todas as características da ferramenta *Rose*, mas sim a utilização de alguns de seus recursos a fim de alcançar o objetivo proposto. Nesse caso, maiores detalhes sobre os demais recursos oferecidos pelo *Add-in* podem ser encontrados em [RAT 2000].

O *Add-in* pode ser desenvolvido para a concretização de várias tarefas. No âmbito desta pesquisa, desenvolver um *Add-in* significa gerar um programa no editor da ferramenta denominado *Rational Rose Script Editor*, que possa ler e interpretar o esquema conceitual criado pelo projetista no módulo diagrama de classes do *Rose* e transcrevê-lo para o *script SAIFTalk*, resultando em um arquivo *SAIF*.

Como a tarefa de gerar um arquivo *SAIF* é realizada inúmeras vezes para as mais diversas aplicações de SIG por diferentes projetistas, torna-se necessário adicionar um submenu à ferramenta, pois é através desse menu que a geração de código *SAIF* é executada automaticamente pelo projetista da aplicação. Assim, o menu torna-se responsável pela conexão entre a interface da ferramenta e o programa desenvolvido no *Script Editor*. A linguagem utilizada por esse editor chama-se *Rose Script* e é considerada uma extensão da linguagem *Summit BasicScript* [RAT 2000].

Além da geração do código *SAIF* e da configuração de menu, é possível a inclusão de uma ajuda *on-line*, que descreve as características relativas ao processo. Outra característica utilizada neste trabalho trata dos esteriótipos que estão inseridos como elementos no diagrama de classes da ferramenta. Esses esteriótipos estão implementados no trabalho, através da definição de um manual desenvolvido por um dos membros do grupo de pesquisa SIGMODA [LIS 2000].

Durante a *Geração de Código SAIF*, algumas decisões são estáticas como, por exemplo, as definições das regras de correspondência, descritas na Seção 3.1.1. Tais regras são executadas pelo programa sem qualquer interferência do projetista. Enquanto que as regras de transformação, definidas na Seção 3.1.2, são semi-dinâmicas, pois algumas decisões são tomadas pelo projetista com auxílio do programa. Essas variantes são apresentadas em forma de perguntas (Figura 4.2) e buscam informações que não estão definidas ou informações que afetam diretamente no resultado do arquivo final.

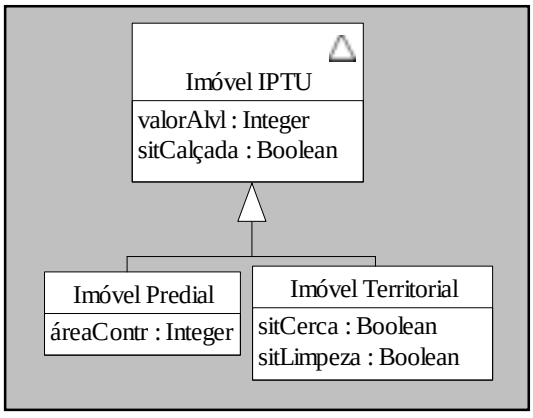
Inicialmente, o programa lê cada classe do esquema, procurando todas as classes que são generalizações e após encontrar a primeira, verifica a existência de associações relacionadas a essa superclasse e solicita o *Id* de cada uma. O *Id* objetiva elaborar o relacionamento entre as futuras tabelas do modelo de dados da aplicação, onde todas as informações declaradas ou selecionadas pelo projetista são salvas em um único *arquivo temporário.*

Após localizar as classes que estão associadas com a superclasse, executa-se o mapeamento, conforme a regra nas associações `1:N´, e solicita-se a decisão do projetista no caso de associações `1:1` e ´n:N`, salvando todas as informações dessas classes (Id´s, atributos, domínios e valores) no arquivo *temporário.* Esse arquivo é constantemente atualizado.

A próxima etapa é solicitar ao projetista o tipo de generalização/especialização que deseja implementar para a superclasse em questão. A solicitação é apresentada ao projetista através de uma tela com a lista de todas as regras (Anexo 3). Dependendo da escolha do projetista, o programa elabora um determinado *script*, ou seja, instancia somente a superclasse ou instancia a subclasse ou ainda, pode instanciar a subclasse e a superclasse. As regras básicas são novamente descritas abaixo:

- Compartilhar o *Id* da superclasse com a subclasse e instanciar ambas, sem considerar o conceito de herança (1);
- Aplicar herança e instanciar somente a subclasse (2);
- Migrar a subclasse para a superclasse e instanciar somente a superclasse (3);

Se a opção 1 for selecionada, o gerador armazena informações sobre a superclasse e a subclasse. Da superclasse armazena o seu *Id* e os atributos definidos na modelagem, e acrescenta mais um atributo, denominado *Type*, o qual objetiva estabelecer qual das subclasses o sistema deve percorrer em uma eventual pesquisa. A respeito da subclasse, o gerador armazena o *Id* da superclasse e demais atributos de cada subclasse especializada. Em relação a utilização dos esteriótipos, torna-se necessário verificar o tipo de situação do diagrama em questão e qual regra está disponível para essa situação, conforme ilustra a Figura 3.6. Tanto as informações dos esteriótipos quanto da superclasse e da subclasse são armazenadas no arquivo *temporário*, conforme demonstra a linguagem *SAIFTalk* a seguir.




```
<GeographicObject
  subclass:     Imóvel IPTU
  attributes:   ID_Imovel       Integer
                valorAlvl       Integer
                sitCalçada      Boolean
                type            Integer
  restricted:   position.geometry:null
  comments:     "este script traduz a
                estrutura da regra 1,
                devido ao objetivo  da
                aplicação"
 >
```

```
<Imóvel IPTU
  subclass:     Imóvel Predial
  attributes:   ID_Imovel       Integer
                áreaContr       Integer
  restricted:   position.geometry:null
  comments:     “essa classe compartilha o Id da classe pai”.
>

<Imóvel IPTU
  subclass:     Imóvel Territorial
  attributes:   ID_Imovel       Integer
                sitCerca        Boolean
                sitLimpeza      Boolean
  restricted:   position.geometry:null
  comments:     “essa classe compartilha o Id da classe pai”.
>
```

No caso da opção 2 ser escolhida pelo projetista, o gerador armazena no arquivo *temporário* as informações relativas ao nome de cada subclasse, *Id´s* e atributos, além dos atributos herdados da superclasse, conforme demonstra o *script* abaixo:

```
<GeograpicObject
subclass:       Imóvel Predial
attributes:     valorAlvl       Integer
                sitCalçada      Boolean
                ID_Imóvel       Integer
                áreaContr       Integer
resctricted:    position.geometriy:null
comments:       “a superclasse é mapeada para cada subclasse”.
>
<GeographicObject
subclass:       Imóvel Territorial
                valorAlvl       Integer
                sitCalçada      Boolena
                ID_Imóvel       Integer
                sitCerca        Boolean
                sitLimpeza      Boolean
restricted:     position.geometry:null
comments:       “a superclasse é mapeada para cada subclasse”.
>
```

Caso o projetista da aplicação selecione a última opção, a situação tornase oposta ao item anterior, visto que todas as informações são descritas diretamente na superclasse, com o acréscimo do atributo *Type*. Todas as informações são armazenadas no arquivo *temporário,* conforme o *script* a seguir:

```
<GeographicObject
  subclass:     Imóvel IPTU
  attributes:   ID_Imovel       Integer
                valorAlvl       Integer
                sitCalçada      Boolean
                áreaContr       Integer
                sitCerca        Boolean
                sitLimpeza      Boolean
                type            Integer
  restricted:   position.geometry:null
  comments:     “este script traduz a estrutura da regra 3, devido ao
                objetivo  da aplicação”
>
```

Após o *Gerador* percorrer todas as generalizações do esquema conceitual e os respectivos relacionamentos associados às superclasses, efetua-se a busca das classes que são agregadas de outras. A agregação é tratada, neste trabalho, como uma associação `1:N`, conforme descrito na Seção 3.1.2.6. Finalmente, são tratadas as classes "solitárias" no esquema, que não são superclasses, subclasses ou associadas a essas.

Como referido na Seção 4.3.1, embora o *tema* apresente correspondência na linguagem *SAIFTalk* (Seção 3.1.1), não é traduzido para essa linguagem, visto que a ferramenta *FME* não o reconhece como parte da estrutura do arquivo.

Portanto a composição final desse arquivo temporário descreve a estrutura básica de cada classe, seus identificadores, atributos, domínios, tamanho dos campos, restrições relativas à representação gráfica de cada classe, associações e generalizações já refinadas durante a elaboração das regras de mapeamento.

4.3.3 Componentes para a Geração de Arquivos *SAIF*

O objetivo dessa Seção é apresentar os componentes que formam um arquivo *SAIF*, pois além do *arquivo temporário*, gerado no módulo *Gerador de Código SAIF*, com base nos esquemas conceituais da aplicação, é necessária a elaboração de outros arquivos.

Esses arquivos são gerados na ferramenta *RationalRose2000* em paralelo com *arquivo temporário*, e sem nenhuma interferência por parte do projetista. A estrutura completa de um arquivo *SAIF*, definido na Figura 4.3, é imprescindível para a utilização da ferramenta de tradução de dados espaciais *FME2000* e, consequentemente, para a definição de um esquema lógico para qualquer software de SIG. Os arquivos são descritos abaixo e estão ilustrados na Figura 4.3. A estrutura de cada um dos arquivos pode ser vista nos Anexos 4 a 8.

O componente *Imports.dir* contém uma lista fixa dos objetos que são importados pelo conjunto de dados *SAIF*.

O *internals.dir* apresenta uma lista fixa dos objetos que são definidos e especificados pelo conjunto de dados *SAIF*. Essa lista não é visível externamente.

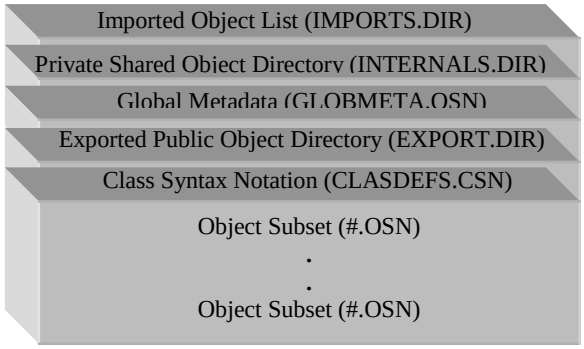


FIGURA 4.3 – *Layout* dos componentes do arquivo SAIF [SAI 94]

O componente *Globmeta.osn* é um conjunto fixo de metadados sobre o próprio *SAIF* como, por exemplo, nome do projetista que está elaborando esse arquivo, data e horário da criação, versão do *SAIF* e outros dados.

Outro componente necessário para a tradução dos modelos de dados é o *Clasdefs.csn*, que deve ser integrado ao *arquivo temporário*, descrito na Seção 4.3.2. Essa união contém a definição de todas as classes do esquema padrão *SAIF* (parte estática), juntamente com a descrição das classes definidas pelo projetista da aplicação (parte dinâmica, criada a cada novo esquema conceitual).

Na parte estática do arquivo *Clasdefs.csn* está descrita toda a estrutura que compõe o padrão *SAIF*. Esse arquivo está elaborado conforme a definição da estrutura da linguagem formal *SAIFTalk*, descrita na Seção 2.2.2.2.

A parte dinâmica sofre constante alteração, pois a cada nova elaboração de um esquema conceitual pelo projetista, baseado no *framework* GeoFrame, esse arquivo, denominado *temporário* é modificado e integrado à parte estática do arquivo formando, então, o arquivo *Clasdefs.csn.*

O componente *Exports.dir* descreve a lista de todos os objetos exportados pelo *SAIF*. Ele é composto por duas classes de objetos externos. Na primeira, estão incluídos os *visible exported objects*, objetos visíveis aos projetistas, mas que não podem ser referidos por outro conjunto de dados. E o segundo tipo de classe são os *sharable exported objects*, que também são visíveis aos projetistas, entretanto podem ser referidos por outra aplicação. Neste trabalho, utiliza-se o primeiro grupo, pois cada classe pertence a uma única aplicação.

O *Exports.dir* é responsável pela exportação das classes de um formato de arquivo para outro. Cada classe a ser mapeada deve ser referida nesse arquivo, o qual apresenta um apontador para um arquivo do tipo *osn* de mesmo nome. Nos arquivos *osn* são descritos os objetos pertencentes à cada classe, conforme descrito abaixo.

Os componentes #.osn são obrigatórios, pois como o *FME* é uma ferramenta de *tradução de dados espaciais*, é necessária a existência de dados. Entretanto, cabe destacar que, em nível conceitual, os esquemas não representam objetos ou instâncias de classes. Porém, o *FME* não permite mapear esquemas de dados sem dados, tornando-se necessária a elaboração de objetos *defaults* para cada classe modelada no esquema, resultando assim, um arquivo *.osn* para cada classe.

Esses *defaults* são criados com a finalidade de concretizar a tradução do arquivo *SAIF* para qualquer outro software de SIG, através da ferramenta *FME*. Os objetos *defaults* são criados pelo programa *Gerador de Código SAIF*, sem qualquer interferência do projetista, porém, cabe a ele, a tarefa de apagar esses dados *defaults* quando acessar o software de SIG. A estrutura desse arquivo é descrito pelo componente *OSN* da linguagem *SAIFTalk* (Seção 2.2.2.2).

Os principais componentes que formam um arquivo *SAIF* são: *clasdefs.csn*, *exports.dir* e os *#.osn*. O arquivo *clasdefs.csn* descreve cada classe do esquema. O *exports.dir* desempenha a função de ligação entre cada classe definida no arquivo *clasdefs.dir* e cada arquivo *#.osn*. Os arquivos *.osn* contêm todas as especificações do objeto e são eles que praticamente descarregam os modelos e os dados na base de dados durante o processo de tradução automática realizada pelo *FME*. A Figura 4.4. ilustra os componentes necessários para formar um arquivo *SAIF*, exigido pela ferramenta *FME*.

Conforme ilustrado na Figura 4.4, as regras de mapeamento, definidas no Capítulo 3, são inseridas no módulo da ferramenta *Rose2000*, com a finalidade de gerar um novo *Add-in* para *SAIF*. Os componentes que formam esse arquivo *SAIF* são descritos na Figura 4.4.

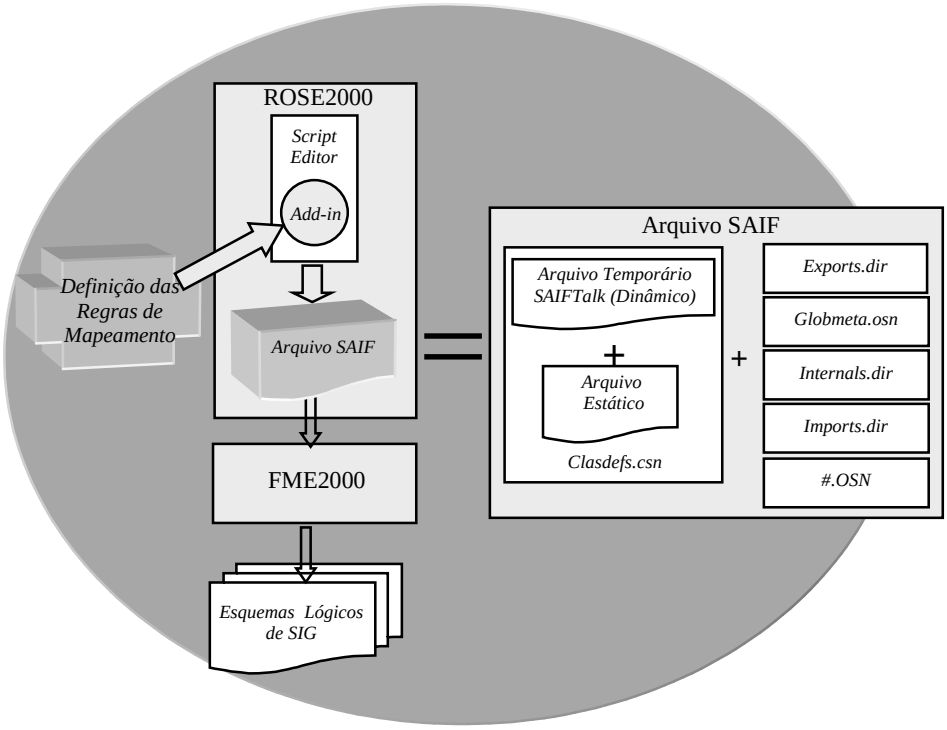


FIGURA 4.4 – Componentes utilizados para gerar modelos lógicos para SIG

O resultado obtido dessa etapa é a união dos cinco componentes citados anteriormente, somados aos arquivos *#.osn*, gerados para cada classe da aplicação. Dessa forma, o módulo de *Geração de Código SAIF* é finalizado e o projetista pode salvar o seu arquivo *SAIF*.

4.3.4 Tradução Automática e a Especificação do Modelo de Dados

Conforme mencionado anteriormente, a ferramenta *FME* realiza tradução automática de dados espaciais. Para tanto, utiliza um arquivo *semantic mapping*, onde estão armazenadas as regras necessárias para traduzir um arquivo de um formato específico de SIG para outro.

Para realizar a tradução do arquivo *SAIF* para qualquer software de SIG na ferramenta *FME*, o projetista deve:

- selecionar o padrão *SAIF* como arquivo de entrada;
- inserir o arquivo gerado (*.*SAF*);
- selecionar o software para onde deseja mapear e um nome para o esquema lógico correspondente.

Logo a seguir, seleciona-se o tipo de arquivo semântico, neste caso, *semantic mapping* para *SAIF*. Conforme o software de SIG selecionado, haverá uma variação no número de arquivos de saída. Essa variação é determinada automaticamente pelo *FME*. No Anexo 10, encontra-se a tela utilizada pela ferramenta *FME* para executar a tradução.

No software selecionado durante a tradução automática, o projetista deve acessar os arquivos gerados, apagar os objetos *default* criados durante o processo de *Geração de Código SAIF*, e iniciar o processo de inserção de instâncias no banco de dados.

4.4 Exemplo de Aplicação das Regras de Mapeamento

Esta Seção apresenta parte de uma aplicação que serve de exemplo para os conceitos desenvolvidos e propostos nos capítulos anteriores. Inicia-se pelo mapeamento de um esquema conceitual na ferramenta *Rose2000*, que foi estendida para transformar os esquemas conceituais em arquivos *SAIF*, conforme a ilustração do Anexo 9.

Para esse exemplo, buscou-se implementar um dos temas relativos ao projeto do SIGPROGB[5], denominado *Hidrografia*, que trata da área de controle ambiental [LIS 2000], conforme ilustra a Figura 4.5.

Conforme ilustra o exemplo da Figura 4.5, cada ponto de amostragem se localiza dentro de um recurso hídrico. Dessa forma, a classe *PtoAmostragem* está associada à classe *RecursoHídrico* que, por sua vez, possui outras associações no contexto do projeto. Cada *RecursoHídrico* se localiza dentro de uma *BaciaHidrográfica*. As classes *Rio* e *Lago* são especializações da classe *RecursoHídrico*.

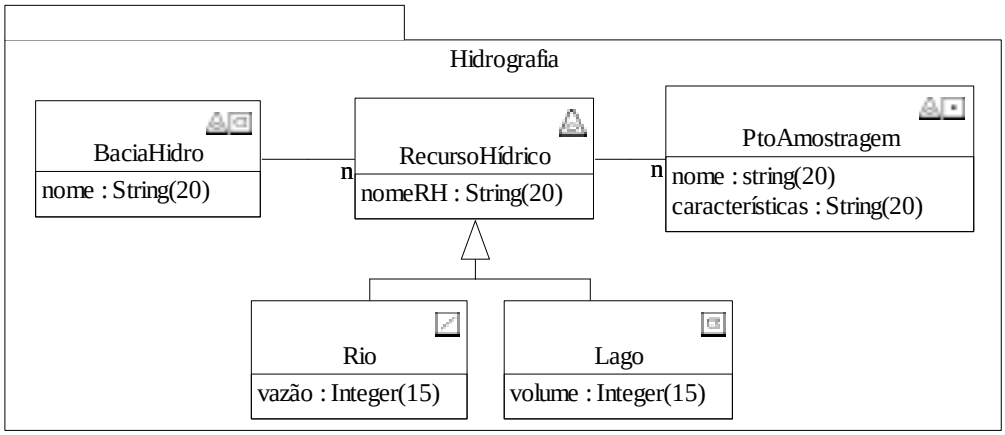


FIGURA 4.5 – Exemplo de aplicação de regras de mapeamento

Os conjuntos de regras de mapeamento, definidos no Capítulo 3, são demonstrados através da realização do mapeamento de um esquema conceitual real, definido a partir

[5] SIGBPROGB envolve a modelagem conceitual do SIG do Programa Pró-Guaíba e tem como objetivo o desenvolvimento de uma série de ações visando a melhoria da qualidade da água em toda a bacia hidrográfica do Rio Guaíba.

do *framework* GeoFrame, para a linguagem *SAIFTalk* do padrão *SAIF*, gerando como resultado o modelo lógico para um determinado SIG.

O *script* principal, onde constam as regras de mapeamento (correspondência e transformação), denominado *arquivo temporário* (Seção 4.3.2), está descrito a seguir, porém os demais arquivos, também necessários e elaborados para esse esquema conceitual, estão descritos no Anexo 11.

```
<GeographicObject
   subclass:    RecursoHídrico::SIGPROGB
   attributes:  ID_RecursoHídrico       Integer(15)
                ID_Bacia                Integer(15)
                NomeRH                  String(20)
   restricted:  position.geometry:null
   comments:    “esta classe contem o ID da classe BaciaHidro, devido
                ao relacionamento 1:N entre esta classe e BaciaHidro”.
 >
<GeographicObject
   subclass:    Rio::SIGPROGB
   attributes:  ID_RecursoHídrico       Integer(15)
                vazão                   Integer(15)
   restricted:  position.geometry:arc
   comments:    “esta classe é maepada para o esquema lógico, devido a
                existência   da   sua   representação   espacial   nas
                subclasses.”
 <GeographicObject
   subclass:    Lago::SIGPROGB
   attributes:  ID_RecursoHídrico       Integer(15)
                volume                  Integer(15)
   restricted:  position.geometry:polygon
   comments:    “esta classe é maepada para o esquema lógico, devido a
                existência da sua representaçãoespacial das
                suvclasses”.
 >
<GeographicObject
   subclass:    BaciaHidro:SIGPROGB
   attributes:  ID_Bacia        Integer(15)
                nome            String(20)
   restricted:  position.geometry:polygon
   comments:    “comments”

 >
<GeographicObject
   subclass:    PtoAmostragem:SIGPROGB
   attributes:  ID_RecursoHídrico       Integer(15)
                ID_PtoAmostragem        Integer(15)
                nome                    String(20)
                características         String(20)
   restricted:  position.geometry:point
   comments:    “esta classe contem o ID da classe RecursoHídrico,
                devido  ao relacionamento 1:N entre esta classe e a
                BaciaHidro”
 >
```

No contexto acima, algumas regras de correspondência e transformação são aplicadas para a definição desse esquema conceitual em linguagem formal *SAIFTalk*. O primeiro conjunto de regras trata da verificação das entidades, que estão envolvidas no esquema como, por exemplo, objeto convencional ou fenômeno geográfico, sendo que a

aplicação dessa regra depende do tipo de esteriótipo que cada classe apresenta. Outra regra desse mesmo conjunto relaciona-se à representação espacial de cada fenômeno geográfico, que também utiliza-se de esteriótipos. Esse tipo de representação espacial é traduzida para a linguagem *SAIFTalk* através da cláusula *restricted*. Entretanto, a representação da classe *tema* não é traduzida para a linguagem *SAIFTalk*, consequentemente, não sendo tratado pelo protótipo apresentado neste trabalho, conforme descrito nas Seções 4.3.1 e 4.3.2.

As regras de transformação aplicadas nesse esquema são basicamente as relativas à generalização e associação. Nesse caso, a regra de generalização baseia-se na construção de uma tabela para cada classe do esquema, devido à utilização de esteriótipos pelas subclasses. Ambas as associações são tratas da mesma maneira, visto que, são definidas na forma de ´1:N´, onde a classe do lado ´N´ recebe o *Id* da classe do lado ´1´.

Com o resultado deste *arquivo temporário* e dos demais especificados no Anexo 11, o *Gerador de Código SAIF* (Anexo 9) fornece um arquivo com extensão *SAF*, o qual é interpretado pela ferramenta *FME2000* (Anexo 10). O resultado da transformação para o modelo de dados do SIG comercial *Geomedia* é um arquivo *shp + dbf + shx* e está ilustrado no Anexo 12.

5 Conclusão

Esta dissertação oferece uma contribuição para sistemas de informação geográfica, através da elaboração de um conjunto de regras genéricas de mapeamento de esquemas conceituais, baseados no *framework* GeoFrame, para os esquemas lógicos dos softwares de SIG. Para tanto, a pesquisa foi abordada a partir do nível conceitual, com a utilização de uma técnica para a criação de esquemas conceituais e com a proposição de uma ferramenta para a criação de diagramas de classe.

Como pôde ser investigado, existe comercialmente uma ferramenta, denominada *FME*, que é capaz de realizar o mapeamento de modelos de dados de um software de SIG para outro, tornando-se desnecessário produzir um novo conjunto de regras de mapeamento para cada software de SIG, ou ainda, realizar uma generalização do esquema lógico de alguns SIG. Porém, o *FME* trabalha a nível lógico e a proposta apresentada neste trabalho busca realizar o mapeamento de esquemas conceituais para esquemas lógicos através da utilização dessa ferramenta e de um padrão de intercâmbio de dados espaciais.

Atualmente, a modelagem de dados em um SIG é direcionada pelas estruturas disponíveis para representar os fenômenos espaciais. Em função disso, cada SIG utiliza uma linguagem própria do modelo de dados implementado. Por exemplo, um mesmo conceito do mundo real é denominado, segundo o SIG que se utiliza, como tema, categoria, camada, plano de informação, cobertura ou mapa. Com isso, torna-se muito difícil a um SIG proporcionar ao projetista um modelo de alto nível para a especificação de aplicações.

Dessa forma, o projetista responsável por uma aplicação de SIG precisa interpretar as necessidades do usuário, mapear o mundo real em esquemas conceituais e transformá-los em dados segundo o modelo implementado pelo SIG. Essa transformação implica na elaboração de regras específicas para o software escolhido pelo usuário. Como não existe um modelo genérico que sirva de interface entre projetistas e alguns modelos proprietários de SIG, torna-se obrigatório elaborar o mapeamento específico para uma determinada ferramenta de SIG.

Partindo desses fatos, investigou-se, neste trabalho, a possibilidade de mapear o esquema conceitual, baseado no GeoFrame, para uma estrutura genérica, onde torna-se possível mapear qualquer esquema conceitual para um padrão de intercâmbio de dados espaciais, através da utilização da sua linguagem formal.

Neste trabalho, utilizou-se a linguagem formal *SAIFTalk*, principalmente pelo fato do padrão *SAIF* permitir a representação das características espaciais consideradas mais relevantes na realidade geográfica e por disponibilizar uma linguagem formal, que pode ser traduzida para os modelos lógicos de alguns SIG.

Considerando essas características, foi elaborado um conjunto de regras genéricas de mapeamento, traduzindo os esquemas conceituais para essa linguagem formal, sendo ela compatível com a estrutura do diagrama de classes apresentado pelo GeoFrame.

O objetivo dessa pesquisa constitui na elaboração de regras para realizar o mapeamento entre os esquemas conceituais, baseados no *framework* GeoFrame, e os modelos lógicos dos software de SIG. Esse objetivo foi alcançado mediante a utilização de uma linguagem disponibilizada pelo padrão *SAIF* e pelo emprego da ferramenta *FME*, tornando-se assim, uma permuta indireta entre os esquemas conceituais e os modelos lógicos dos softwares de SIG.

Um segundo ponto abordado nesta pesquisa, trata da definição de um ambiente de suporte, que permite validar a aplicabilidade das regras de mapeamento em um ambiente real de modelagem de dados geográficos. Essa automatização foi realizada, através da utilização de uma ferramenta de modelagem de dados convencional (*Rose2000*), a qual foi estendida com a utilização de alguns recursos de programação, e de uma ferramenta de tradução de dados espaciais (*FME2000*).

Durante a pesquisa foram constatadas algumas limitações relativas ao ambiente de suporte para mapeamento, mais especificamente sobre a ferramenta *FME*.

A primeira limitação, refere-se à execução de traduções automáticas realizadas pelo *FME*. Como o objetivo dessa ferramenta é realizar traduções de dados espaciais de um software de SIG para outro, à nível lógico, torna-se obrigatória a presença dos dados associados ao modelo de dados. Dessa forma, a solução empregada foi gerar, automaticamente, dados *default* para cada classe da aplicação durante o processo de *geração de código SAIF*.

A segunda limitação, refere-se a utilização da estutura da classe *SpatialDataSet*. Essa estrutura não é reconhecida pelo *FME*, não sendo assim traduzida para a linguagem *SAIFTalk*, embora apresente correspondência nessa linguagem.

Um item que interfere na modelagem conceitual das classes com diferentes representações espaciais, é a falta de especificação de uma cláusula na linguagem *SAIFTalk*, que possibilite tratar essa situação. Nesse caso, aconselha-se ao projetista definir apenas um tipo de representação para cada fenômeno geográfico da aplicação.

Além disso, durante o mapeamento das associações e generalizações, foi necessária a utilização de chaves primárias para realizar a materialização dos relacionamentos. Entretanto, como em orientação a objetos o conceito de chave primária não é explicitamente declarado [RUM 91], a alternativa encontrada foi, durante a *geração do código SAIF*, solicitar ao projetista a definição de um identificador para cada classe. Dessa forma, tornou-se possível realizar o mapeamento das associações e das generalizações entre as classes da aplicação.

As soluções encontradas tanto para as limitações da ferramenta *FME* quanto para a linguagem *SAIFTalk*, possibilitaram a adequação das regras de mapeamento, obtendo como resultado a proposta de uma arquitetura de suporte para mapeamento e um protótipo para aplicação das regras definidas.

Durante a pesquisa foi encontrada uma ferramenta denominada *Perceptory* que também trabalha no mesmo nível de transformação de esquemas conceituais para lógicos. Entretanto, essa ferramenta define regras de mapeamento de acordo com a definição *PLVs* (*Plug-in for Visual Language*), especificas para um software de SIG.

Assim, tanto o *Gerador de Código SAIF* quanto a ferramenta *Perceptory* procuram tratar do mesmo problema, porém a partir de modelos conceituais diferenciados. Enquanto que o *Gerador de Código SAIF* trata dos esquemas de modelagem a partir do GeoFrame e busca definir alguns modelos lógicos de SIG comerciais, o *Perceptory* utiliza o conjunto de conceitos e construtores, denominado *PLVs* para especificar os seus próprios esquemas conceituais, limitando-se a tradução para um único modelo lógico.

Através desta pesquisa, bem como da bibliografia utilizada, este documento poderá ser uma fonte de apoio para trabalhos futuros. Alguns destes trabalhos podem, por exemplo, possibilitar a extensão de regras de mapeamento, relativo aos aspectos temporais, à qualidade de dados e aos relacionamentos espaciais, os quais são possíveis tanto com base no GeoFrame como com base em *SAIF*.

Além disso, seria possível incluir uma metodologia de engenharia reversa de aplicações de SIG, gerando-se esquemas GeoFrame, a partir do ambiente de suporte desenvolvido e da utilização do módulo proposto (*Gerador de Código SAIF*).

Anexo 1 Gramática BNF/CSN

As linguagens de programação podem ser especificadas pela descrição de sua estrutura (sintaxe de linguagem). Para especificar essa sintaxe, o padrão SAIF optou pela notação denominada gramática livre de contexto ou BNF.

Além de especificar a sintaxe da linguagem, essa gramática pode ser utilizada como auxílio para guiar a tradução de programas. O objetivo geral da BNF/CSN é fornecer, além de uma sintaxe, um modelo para novas construções baseadas nessa estrutura.

<schemaDefinition> ::= <definition> | <definition> <schemaDefinition>
<definition> ::= <openDefinition><definitionBody> <closeDefinition>
<definitionBody> ::= <abstractObjectDefinition>|<enumerationDef>|
<domainDefinition>
<abstractObjectDefinition> ::= <parentName> <subclass>
[<tag>]
[<subclassing>]
[<attributes>]
[<classAttributes>]
[<defaults>]
[<restricted>]
[<constraints>]
[<classAttributeDefaults>]
[<classAttributeValues>]
[<comments>]
<parentName> ::= <className>{<comma>[<whitespace>] className>}
<className> ::= <upperName> [<schemaID>]
<schemaID> ::= "::" <upperName>
<subclass> ::= "subclass:" <className>
<tag> ::= "tag:" <tagCategory> <tagNumber>
<tagCategory> ::= "universal" | "application" | "private"
<tagNumber> ::= <unsignedInteger>
<attributes> ::= "attributes:" <attributeDef> {<attributeDef>}
<attributeDef> ::= <attribNameSpec> [<forceType>] <attributeType>
<attributeType> ::= <className> | <stringType> |
(<collection> "(" [<forcetype>] <className>
| <stringType> ")")
<stringType> ::= "String" ["(" <unsignedInteger> ")"]
<collection> ::= "List" | "Set" | "Relation"
<attribNameSpec> ::= <attributeName> | <optionalAttrib>
<attributeName ::= <lowerName>
<openValueSpec> ::= <openParen>
<closeValueSpec> ::= <closeParen>
<openOptional> ::= <openBracket>
<closeOptional> ::= <closeBracket>

```
<optionalAttrib>          ::=    <openOptional> <attributeName> <closeOptional>
<subclassing>             ::=    "subclassing:" <textString>
<defaults>                ::=    "defaults:" <defaultSpec> {<defaultSpec>}
<defaultSpec>             ::=    <attributeName> <setTo> <value>
<value>                   ::=    <textString> | <number> | <enumtag>
<number>                  ::=    <float> | <integer>
<enumTag>                 ::=    <lowerCaseLetter> | <poundSign>
                                 { <character> | <dash> }
<restricted>              ::=    "restricted:" <restrictionSpec> {<restrictionSpec>}
<restrictionSpec>         ::=    <valueRestriction> | <typeRestriction>
<valueRestriction>        ::=    <attribPathName> <setTo> <valueChoice>
<valueChoice>             ::=    <valueSpec> {<chooser> <valueChoice>}
                                 <valueSpec> ::= <value> | <range>
<range>                   ::=    <lowerBound> ".." <upperBound>
<lowerBound>              ::=    <integer>
<upperBound>              ::=    <integer>
<attribPathName>          ::=    {<attribNameSpec>[<collectionMember>]<selector>}
                                 <attribNameSpec>
<collectionMember>        ::=    "{}"
<typeRestriction>         ::=    <typePathName> <setTo> <typeChoice>
<typeChoice>              ::=    <typeSpec> {<chooser> <typeChoice> }
<typeSpec>                ::=    [ <excludeType> | <forceType> ] <className>
<typePathName>            ::=    [<attribPathName>] (<wildcard> |<selector> )
                                 <className>
<constraints>             ::=    "constraints:" <textString>
<comments>                ::=    "comments:" <textString>
<classAttributeValue>     ::=    { <attributeName> <setTo> <value> }
<classAttributeValues>    ::=    "classAttributeValues:"<classAttributeValue>
                                   {< classAttributeValue>}
<classAttributeDefaults> ::=     "classAttributeDefaults:"<classAttributeValue>
{<classAttributeValue>}
<enumerationDef>          ::=    "Enumeration"
                                 <subclass>
                                 [<tag>]
                                 <enumTags>
                                 [<comments>]
<enumTags>                ::=    "values:" <tagSpec> {<tagSpec>}
<tagSpec>                 ::=    <enumTag> [<numericRep>]
<numericRep>              ::=    <openValueSpec><unsignedInteger> <
                                 <closeValueSpec>
<domainDefinition         ::=    "Domain"
                                 <subclass>
                                 [<tag>]
                                 <primitiveType>
                                 <domainValues>
                                 [<comments>]
<primitiveType>           ::=    "primitiveType:" <className>
```

<domainValues> *::=* *"values:" (<strings> | <integers | <floats>)*
<strings> *::=* *<textString> {<textString> }*
<floats> *::=* *<float> {<float> }*
<integers> *::=* *<integer> {<integer> }*

Anexo 2 Gramática BNF/OSN

Esta gramática é bem mais compacta que a anterior, porém é repetida, várias vezes, durante a definição das instâncias de classes.

<instances> *::=* *<abstractObject> {<abstractObject>}*

<expression> *::=* *<integer>* |
<float> |
<textString> |
<bitString> |
<octetString> |
<enumTag> |
"true" |
"false" |
"nil" |
<abstractObject> |
<list> |
<set> |
<relation> |
<identifier>

<attribute> *::=* *<lowerName> <setTo> <expression>*

<attributeList> *::=* *<attribute> {<attribute>}*

<expressionList> *::=* *<expression> {<expression>}*

<abstractObjectList> *::=* *<abstractObject> {<abstractObject>}*

<openAbstractObject> ::= *<openParen>*

<closeAbstractObject> ::= *<closeParen>*

<openList> *::=* *"List" <openBrace>*

<closeList> *::=* *<closeBrace>*

<openSet *::=* *"Set" <openBrace>*

<closeSet> *::=* *<closeBrace>*

<openRelation> ::= *"Relation" <openBrace>*

<closeReation> ::= *<closeBrace>*

<abstractObject> *::=* *<upperName> <openAbstractObject> <attributeList>*
<closeAbstractObject>

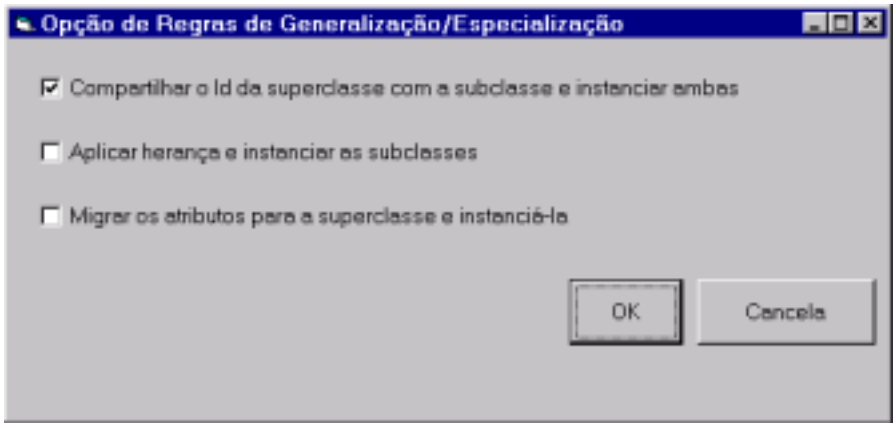
<list> *::=* *<openList> <expressionList> <closeList>*

<set> *::=* *<openSet> <expressionList> <closeSet>*

<relation> *::=* *<openRelation><abstractObjectList><closeRelation>*

<identifier> ::= "*" (<integer> | <textString>)

Anexo 3 Tela de seleção de regras de generalização



Anexo 4 Estrutura do arquivo imports.dir

```
<AbstractObject
      subclass: ImportedObjects
      attributes: handles Set (ImportedObjectHandle)
```

ImportedObjectHandle

```
<AbstractObject
    subclass:   ImportedObjectHandle

    attributes: referenceID             String
                type                    String
                datasetLocator          String
```

Anexo 5 Estrutura do arquivo internals.dir

```
<AbstractObject
      subclass: InternallyReferencedObjects
      attributes: handles Set(ObjectHandle)
```

ObjectHandle

```
<AbstractObject
    subclass:   ObjectHandle

    attributes: referenceID     String
        objectSubset    String
        offset          Integer32Unsigned
```

Anexo 6 Estrutura do arquivo globmeta.csn

```
<AbstractObject
    subclass:   GlobalMetadata

    attributes: saifRelease             String
                saifProfile             Profile
                [userProfile]           UserProfile
                [creationTime]          TimeStamp
                [toolkitVersion]        String
                [remarks]               String
```

Profile

```
<AbstractObject
    subclass:   Profile

    attributes: authority       String
        idName  String
        version         String
        [description]   String
```

UserProfile

```
<AbstractObject
    subclass:   UserProfile

    attributes: [coordDefs]     LocationalDefinitions
                [characters]    CharDef
                [organization]  Numeric
```

LocationalDefinitions

```
<AbstractObject
    subclass:   LocationalDefinitions

    attributes: c1              NumericInstanceType
                c2              NumericInstanceType
                c3              NumericInstanceType
```

CharDef

```
<AbstractObject
    subclass:   CharDef

    attributes: standard        String
                [version]       String
                [private]       Boolean
                [code]          String

    defaults:   private:false
```

Anexo 7 Estrutura do arquivo clasdef.csn

```
<PrimitiveType
subclass: String
>
<PrimitiveType
subclass: OctetString
>
<PrimitiveType
subclass: BitString
>
<PrimitiveType
subclass: Boolean
>
<PrimitiveType
subclass: Integer
>
<PrimitiveType
subclass: Integer8
>
<PrimitiveType
subclass: Integer16
>
<PrimitiveType
subclass: Integer32
>
<PrimitiveType
subclass: Integer64
>
<PrimitiveType
subclass: Real
>
<PrimitiveType
subclass: Real32
>
<PrimitiveType
subclass: Numeric
>
<PrimitiveType
subclass: Primitive
>
<PrimitiveType
subclass: Type
>
<PrimitiveType
subclass: AbstractObject
>
<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
>
<Coverage
subclass: Field
restricted: position.geometry: VectorSurface | Grid
>
<Field
subclass: IrregularDEM
restricted: position.geometry: MeasuredSurface
position.geometry.masspoints{}: ^Point | DEMpoint
>
<Coverage
subclass: Raster
attributes: contentValues Type
[indexScheme] IndexScheme
defaults: indexScheme: sequentialArray
restricted: position.geometry:Grid
contentValues:ExternalReference | BitString | OctetString
>
<Raster
subclass: GeneralRaster
attributes: channels List(Channel)
defaults: position.geometry: ImageGrid2D
restricted: indexScheme: sequentialArray | interleavedArray
>
<Raster
subclass: CategoricalRaster
attributes: category List(Category)
[instanceType] PrimitiveInstanceType
[indexValueType] PrimitiveInstanceType
defaults: position.geometry: CellGrid2D
restricted: indexScheme: sequentialArray | runLengthEncoded |
nTree
instanceType: integer8 | integer8Unsigned | integer16 |
integer16Unsigned |
integer32 | integer32Unsigned | integer64 | integer64Unsigned
indexValueType: integer8 | integer8Unsigned | integer16 |
integer16Unsigned |
integer32 | integer32Unsigned | integer64 | integer64Unsigned
>
<Raster
subclass: RasterDEM
attributes: [instanceType] NumericInstanceType
restricted: contentValues: ExternalReference | OctetString
>
<GeographicObject
subclass: Graph
attributes: connections Set(ConnectedTo)
>
<Graph
subclass: AcyclicNetwork
>
<Graph
subclass: SingleLineNetwork
restricted: connections{}.objectA.position.geometry:VectorLine
```

```
connections{}.objectB.position.geometry:VectorLine
>
<AbstractObject
subclass: SpatialObject
attributes: [geometry] GeometricObject
[spatialReferencing] SpatialReferencing
>
<AbstractObject
subclass: TemporalObject
attributes: [timeObject] TimeObject
[temporalReferencing] TemporalReferencing
>
<SpatialObject, TemporalObject
subclass: SpatiotemporalObject
>
<AbstractObject
subclass: Annotation
attributes: [textOrSymbol] TextOrSymbolObject
[spatialReferencing] SpatialReferencing
>
<AbstractObject
subclass: GeometricObject
attributes: [qualifier] LocationalQualifier
>
<GeometricObject
subclass: GeometricAggregate
attributes: objects Set(GeometricObject)
restricted: objects{}: Point |
VectorLine |
VectorArea |
VectorVolume |
CellularStructure
>
<GeometricObject
subclass: Point
attributes: [coords] Coordinate
>
<Point
subclass: AlignedPoint
>
<Point
subclass: VectorAtPoint
>
<VectorAtPoint
subclass: VelocityAtPoint
>
<Point
subclass: DEMpoint
>
<Point
subclass: CorrelationPoint
>
<Point
subclass: GeoidPoint
>
<GeoidPoint
subclass: GeoidPointWithError
>
<Point
subclass: PointWithAttitude
```

```
>
<GeometricObject
subclass: VectorLine
>
<VectorLine
subclass: Arc
attributes: pointList List(Point)
>
<Arc
subclass: ArcDirected
>
<ArcDirected
subclass: Vector
>
<Arc
subclass: Segment
>
<VectorLine
subclass: OrientedArc
>
<VectorLine
subclass: Breakline
>
<VectorLine
subclass: Isoline
attributes: value Numeric
arc Arc
[form] IsolineForm
>
<Isoline
subclass: Contour
>
<VectorLine
subclass: Path
>
<VectorLine
subclass: OrientedPath
>
<GeometricObject
subclass: VectorArea
>
<VectorArea
subclass: BoundedArea
attributes: [boundary] Set(VectorLine)
>
<VectorArea
subclass: Polygon
>
<Polygon
subclass: PolygonWithHoles
>
<VectorArea
subclass: Triangle
>
<VectorArea
subclass: VectorSurface
>
<VectorSurface
subclass: SurfacePositions
>
```

```
<VectorSurface
subclass: TIN
>
<TIN
subclass: TINEnvelope
>
<VectorSurface
subclass: MeasuredSurface
>
<VectorSurface
subclass: Isolines
>
<Isolines
subclass: Contours
>
<VectorSurface
subclass: Plane
>
<VectorSurface
subclass: GeneralParametric
>
<GeometricObject
subclass: VectorVolume
>
<VectorVolume
subclass: TIN3D
>
<TIN3D
subclass: TIN3DComplex
>
<VectorVolume
subclass: Tetrahedron
>
<VectorVolume
subclass: TetrahedronComplex
attributes: tetrahedra Set(Tetrahedron)
>
<GeometricObject
subclass: CellularStructure
>
<CellularStructure
subclass: Grid
attributes: [positionOfOrigin] Point
>
<Grid
subclass: PointGrid
>
<PointGrid
subclass: DEMGrid
>
<Grid
subclass: CellGrid2D
>
<CellGrid2D
subclass: BoxGrid
>
<Grid
subclass: ImageGrid2D
>
<Grid
```

```
subclass: CellGrid3D
>
<Grid
subclass: ImageGrid3D
>
<Grid
subclass: CellGrid2DT
>
<Grid
subclass: ImageGrid2DT
>
<Grid
subclass: CellGrid3DT
>
<Grid
subclass: ImageGrid3DT
>
<Grid
subclass: HexagonCellGrid
>
<Grid
subclass: TriangleCellGrid
>
< CellularStructure
subclass: LocationsInGrid
>
<AbstractObject
subclass: TimeObject
>
<TimeObject
subclass: TimeAggregate
>
<TimeObject
subclass: DateTime
>
<DateTime
subclass: Date
attributes: [year] Integer
[month] Integer
[day] Integer
restricted: month: 1..12
day: 1..31
>
<DateTime
subclass: Time
attributes: hour Integer
minute Integer
second Real
restricted: hour: 0..24
minute: 0..59
>
<Date, Time
subclass: TimeStamp
>
<TimeObject
subclass: Interval
>
<Interval
subclass: YearMonthInterval
attributes: [years] Integer
```

```
[months] Integer
defaults: years:0
months:0
>
<Interval
subclass: DayTimeInterval
attributes: [days] Integer
[hours] Integer
[minutes] Integer
[seconds] Real
defaults: days:0
hours:0
minutes:0
seconds:0.0
>
<TimeObject
subclass: Duration
attributes: startTime DateTime
lengthOfTime Interval
>
<AbstractObject
subclass: Metadata
>
<Metadata
subclass: SpatialReferencing
attributes: [coordSystem] CoordinateSystem
[hydrographic] HydrographicReference
[control] Control
[transform] Transform
[positioningMethod] PositioningMethod
>
<Metadata
subclass: TemporalReferencing
attributes: type TimeType
[offset] UTCOffset
[likelyTimeError] Interval
>
<Metadata
subclass: GeneralLocation
attributes: [range] BoundingBox
[tiles] List(Tile)
[informal] String
>
<Metadata
subclass: Quality
attributes: [structuring] Structuring
[positional] Set(PositionalAccuracy)
[attribute] Set(AttributeAccuracy)
[clipping] Boolean
[integrity] Integrity
[remarks] String
>
<Metadata
subclass: Lineage
attributes: [dataSource] OriginalDataSource
[compilationHistory] List(ActionHistory)
[processingHistory] List(ActionHistory)
[remarks] String
>
<Metadata
```

```
subclass: Source
attributes: [terminus] Set(Terminus)
[database] DbInfo
[project] Set(Project)
[releasibility] String
[legality] Legal
[security] Security
[remarks] String
>
<Metadata
subclass: ProductDescription
attributes: [description] String
[products] Set(Product)
[remarks] String
>
<Metadata
subclass: UpdateOperation
attributes: operation UpdateType
linkName String
[effective] DateTime
[remarks] String
>
<AbstractObject
subclass: Coordinate
>
<Coordinate
subclass: CoordT
attributes: t TimeObject
restricted: t: DateTime | Interval
>
<Coordinate
subclass: Coord1D
attributes: c1 Numeric
>
<Coord1D, CoordT
subclass: Coord1DT
>
<Coordinate
subclass: Coord2D
attributes: c1 Numeric
c2 Numeric
>
<Coord2D, CoordT
subclass: Coord2DT
>
<Coordinate
subclass: Coord3D
attributes: c1 Numeric
c2 Numeric
c3 Numeric
>
<Coord3D, CoordT
subclass: Coord3DT
>
<AbstractObject
subclass: ArcDefinition
attributes: type CurveChoice
[supportingPoints] List(Point)
>
<AbstractObject
```

```
subclass: DistanceAndAzimuth
attributes: distance Real
azimuth Real
>
<DistanceAndAzimuth
subclass: DistanceAzimuthAndDeclination
attributes: declination Real
>
<AbstractObject
subclass: DistanceAlong
attributes: distance Real
from StartOrEnd
>
<AbstractObject
subclass: GridReference
attributes: [ordering] GridOrder
[originCorner] OriginReference
[cellPositionAtCorner] Boolean
[originCoordinates] List(Integer)
defaults: ordering:rowOrder
cellPositionAtCorner:true
>
<AbstractObject
subclass: GridFramework
attributes: [gridDimensions] List(Integer)
[gridSpacing] List(Numeric)
[offsetStructure] OffsetStructure
[units] List(UnitMeaning)
defaults: offsetStructure:none
>
<AbstractObject
subclass: CellGeometry
attributes: [cellSize] List(Numeric)
[units] List(UnitMeaning)
[cellShape] CellShape
defaults: cellShape:rectangle
>
<AbstractObject
subclass: TypeDefinition
attributes: type TypeIdentifier
[tag] Tag
>
<TypeDefinition
subclass: AbstractObjectDefinition
attributes: [superclass] List(TypeIdentifier)
[subclassing] String
[attributes] List(AttributeDefinition)
[classAttributes] List(AttributeDefinition)
[defaults] List(RestrictedValue)
[restricted] List(Restricted)
[constraints] String
[classAttributeDefaults] List(RestrictedValue)
[classAttributeValues] List(Restricted)
[comments] String
defaults: superclass.typeName:"AbstractObject"
restricted: classAttributeValues{}:RestrictedValue |
RestrictedValueSet |
RestrictedValueRange
>
```

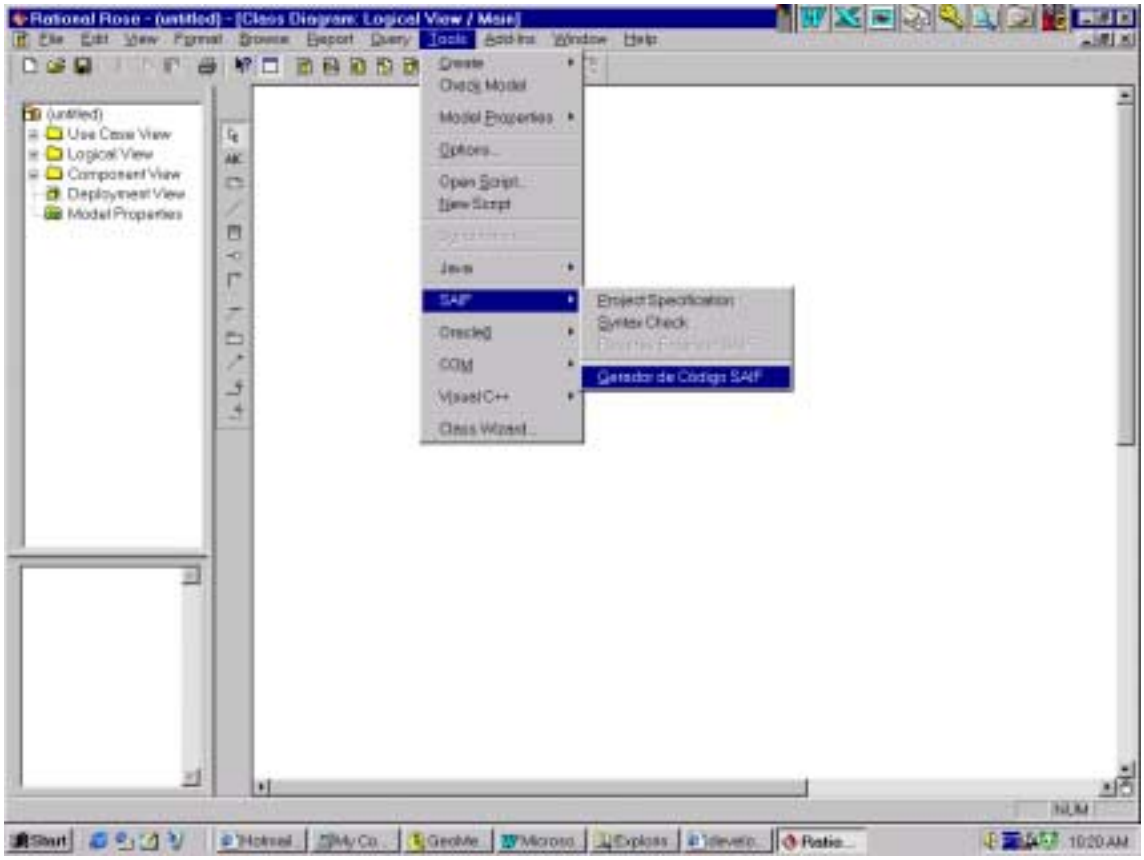
Anexo 8 Estrutura do arquivo exports.dir

```
<AbstractObject
    subclass:   ObjectHandle

    attributes: referenceID     String
        objectSubset    String
        offset          Integer32Unsigned
```

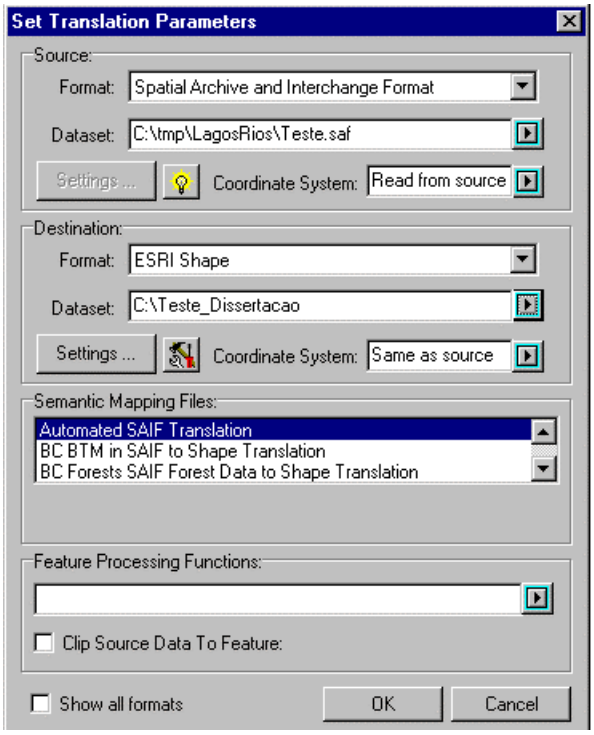
ObjectHandle

```
<ObjectHandle
    subclass:   ExportedObjectHandle

    attributes: type            String
        [sharable]      Boolean

    defaults:   sharable: false
```

Anexo 9 Ferramenta *RationalRose2000* - estendida



Anexo 10 Ferramenta de tradução de dados espaciais - *FME2000*



Anexo 11 Arquivos elaborados a partir de um exemplo de mapeamento

Arquivo Exports.dir

```
ExportedObjects(handles:Set{ExportedObjectHandle(referenceID:"GlobalMe
tadata"
type:"GlobalMetadata"
objectSubset:"globmeta.osn"
offset:0
sharable:false
)
ExportedObjectHandle(referenceID:"Lago"
type:"LagoComposite::MYSCHEMA"
objectSubset:"Lago.osn"
offset:0
sharable:false
)
ExportedObjectHandle(referenceID:"Rio"
type:"RioComposite::MYSCHEMA"
objectSubset:"Rio.osn"
offset:0
sharable:false
)
ExportedObjectHandle(referenceID:"PtoAmostragem"
type:"PtoAmostragemComposite::MYSCHEMA"
objectSubset:"PtoAmostragem.osn"
offset:0
sharable:false
)
ExportedObjectHandle(referenceID:"RecursoHídrico"
type:"RecursoHídricoComposite::MYSCHEMA"
objectSubset:"RHid.osn"
offset:0
sharable:false
)
ExportedObjectHandle(referenceID:"BaciaHidro"
type:"BaciaHidroComposite::MYSCHEMA"
objectSubset:"Bacia.osn"
offset:0
sharable:false
)
})
```

Arquivo GlobalMetadata.osn

```
GlobalMetadata(objectIdentifier:"GlobalMetadata"
creationTime:TimeStamp(day:4
hour:9
minute:19
month:10
second:54
year:2000
)saifProfile:Profile(authority:"Andreia Castro Costa"
idName:"SAIF"
version:"Release 1.1"
)saifRelease:"SAIF 3.2"
```

```
toolkitVersion:"SAIF Toolkit Version 1.3.8 (Aug 08, 1996)"
userProfile:UserProfile(coordDefs:LocationalDefinitions(c1:real64
c2:real64
c3:real64
)organization:4
))
```

Arquivo Lago.osn

```
LagoComposite::MYSCHEMA(objectIdentifier:"Lago"
geoComponents:Set{Lago::MYSCHEMA
(position:SpatialObject
(geometry:Polygon(boundary:Set{Arc(pointList:List
{Point(coords:Coord2D(c1:500000.001 c2:5807740.381))
Point(coords:Coord2D(c1:500000.001
c2:5805528.871))Point(coords:Coord2D(c1:500566.981
c2:5805528.901))Point(coords:Coord2D(c1:501133.951)}
)}))
cdRH_ID: 0
volume:0
)})
```

Arquivo Rio.osn

```
RioComposite::MYSCHEMA(objectIdentifier:"Rio"
geoComponents:Set{Rio::MYSCHEMA
(position:SpatialObject
(geometry:Arc(pointList:List
{Point(coords:Coord2D(c1:560200.000 c2:555400.000))
Point(coords:Coord2D(c1:560401.000 c2:555712.001))}
))
cdRH_ID:0
vazão:0
)})
```

Arquivo PtoAmostragem.osn

```
PtoAmostragemComposite::MYSCHEMA(objectIdentifier:"PtoAmostragem"
geoComponents:Set{PtoAmostragem::MYSCHEMA
(position:SpatialObject
(geometry:point(pointList:List
{Point(coords:Coord2D(c1:000000.000 c2:000000.000))}
))
iDPonto_ID:0
cdRH_ID:0
características:"DefaultCaracterísticas"
)})
```

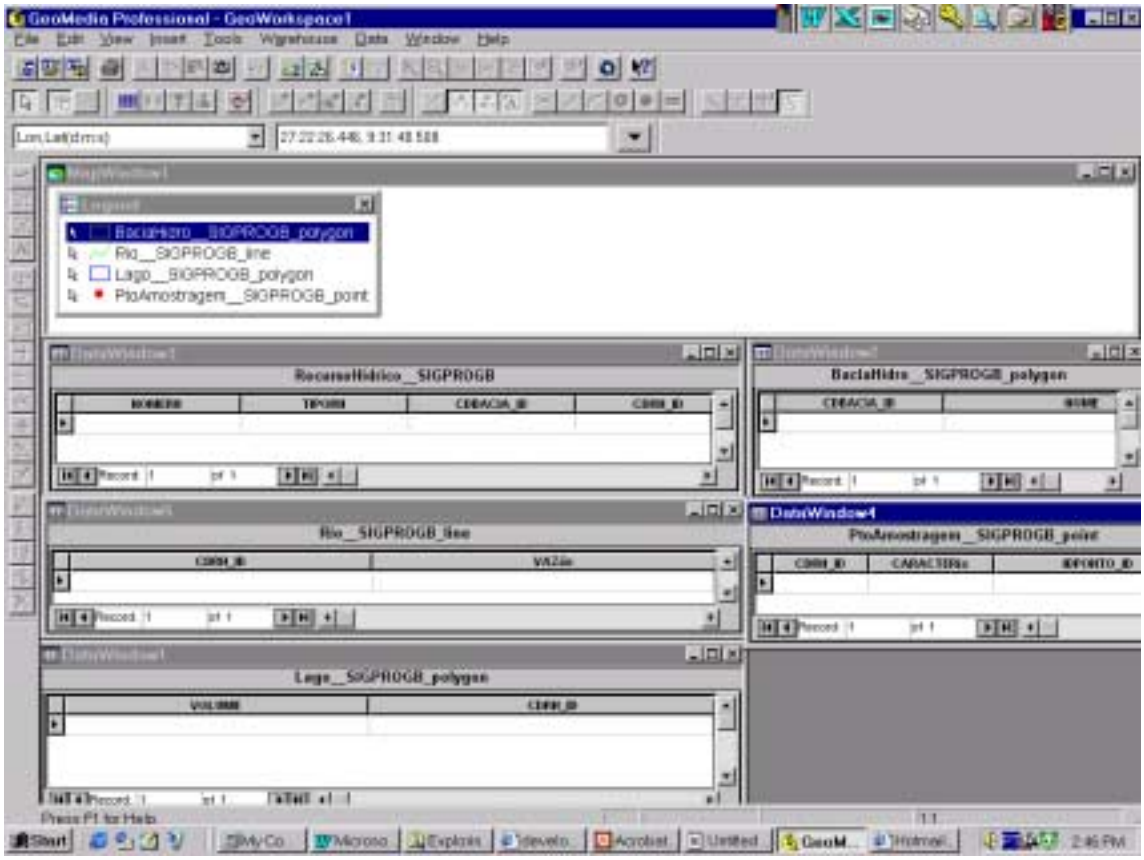
Arquivo RecursoHídrico.osn

```
RecursoHídricoComposite::MYSCHEMA(objectIdentifier:"RecursoHidrico"
geoComponents:Set{RecursoHídrico::MYSCHEMA
(position:SpatialObject
(geometry:point(pointList:List
{Point(coords:Coord2D(c1:000000.000 c2:000000.000))}
))
cdRH_ID: 0
cdBacia_ID: 0
nomeRH:"defaultNome"
tipoRH:0
)})
```

Arquivo Bacia.osn

```
BaciaHidroComposite::MYSCHEMA(objectIdentifier:"BaciaHidro"
geoComponents:Set{BaciaHidro::MYSCHEMA(position:SpatialObject
(geometry:Polygon(boundary:Set{Arc(pointList:List
{Point(coords:Coord2D(c1:500000.001 c2:5807740.381))
Point(coords:Coord2D(c1:500000.001
c2:5805528.871))Point(coords:Coord2D(c1:500566.981
c2:5805528.901))Point(coords:Coord2D(c1:501133.951)}
)}))
cdBacia_ID:0
nome:"defaultNome"
)})
```

Anexo 12 Modelo lógico do *Geomedia* a partir de um esquema conceitual



Bibliografia

[ABR 98] ABRANTES, M.G. **Contribuições para uma metodologia de desenvolvimento de SIG**. Lisboa: IST/UTL, 1998. Tese de Doutoramento.

[AMB 2000] AMBLER, S. Mapping Objects to Relational Database. Disponível em: <http://www-106.ibm.com/developerworks/library/mapping-to-rdb/index.html> Acesso em: 19 maio 2000.

[BAT 92] BATINI, C.; CERI, S.; NAVATHE, S. B. **Conceptual database design**: An Entity Relationship Approach. Redwood, CA: Benjamin/Cummings, 1992.

[BED 89] BÉDARD, Y.; PAQUETTE, F. Extending entity/relationship formalism for spatial information systems. In: AUTOCARTO, 9., 1989. **Proceedings...** [S.1.:s.n.], 1989. p.818-828.

[BED 96] BÉDARD, Y.; MAAMAR, Z. et al. Adapting data models for the design of spatial–temporal databases. **Computer Environmemt and Urban Systems on International Journal**, [S.l.], v.20, n.1, 1996.

[BED 99] BÉDARD, Y. Visual Modelling of spatial databases towards spatial extensions and UML. **Geomatica**, [S.l], v.53, n. 2, p. 169-186, 1999.

[BED 99a] BÉDARD, Y. Principles of spatial databases analyis and design. In: Longley A. P., Goodchild M. et al (Eds.). **Geographical Information System:** principles, techniques, application and managements. 2nd ed. New York: Wiley, 1999. p. 413-424.

[BED 2000] BÈDARAD, Y. **Developments to come in the next version of Perceptory 2000**: Perceptory 2000 - User Guide Topic. Disponível em: <http://www.sirs.scg.ulaval.ca/perceptory/english/news_e.htm>. Acesso em: 25 agosto 2000.

[BRO 2000] BRODEUR, J.; BÉDARD, Y. et al. Modelling geospatial application database using uml-based repositories aligned with international standards in geomatics. **ACMGIS**, Washington DC, v.131, n. 1, 2000.

[BOO 98] BOOCH, G.; JACOBSON, I.; RUMBAUGH, J**. The unified modeling language user guide**. [S.l.]: Addison-Wesley, 1998.

[CAM 94] 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.

[CLE 94] CLEMENTINI, E. ; FELICE, P.D. Object-oriented modeling of geographic data. **Journal of the American Society for Information Science**, [S.1.], v.45, n.9, p.694-704, 1994.

[DAV 99] DAVIS Jr., C. A. Múltiplas representações em bancos de dados geográficos. In: GIS BRASIL CONGRESSO E FEIRA PARA USUÁRIOS DE GEOPROCESAMENTO DA AMÉRICA LATINA, 5., 1999, Salvador. **Anais...** Salvador: Fator GIS, 1999.

[DAV 2000] DAVIS Jr., C. A. **Múltiplas representações em sistemas de informação geográficos**. Belo Horizonte, MG:UFMG, 2000. Tese de Doutorado.

[COA 91] COAD, P.; YOURDON, E. **Object_oriented analysis**. New Jersey: Prentice-Hall, 1991.

[COS 99] COSTA, A.C.; IOCHPE, C**. Definição de regras para mapeamento de esquema conceitual para esquema lógico de SIG**. Porto Alegre: PPGC da UFRGS, 1999.(TI - 932)

[COS 2000] COSTA, A.C.; IOCHPE, C. **Um guia para o modelo SAIF de intercâmbio e armazenamento de dados espaciais**. Porto Alegre: PPGC da UFRGS, 2000 (TI - 800)

[EGE 87] EGENHOFER, M. J.; FRANK, A. U. Object-oriented databases: database requirements for GIS. In: INT. GEOGRAPHIC INFORMATION SYSTEMS SYMPOSIUM, 1987. Arlington. **Proceedings...** Virginia: [s.n], 1987.

[EGE 95] EGENHOFER, M. J. Object-oriented GIS. In: III SIMPÓSIOP BRASILEIRO DE GEOPROCESSAMENTO,3., 1995. **Anais...** São Paulo: USP,1995.

[FUR 98] FURLAN, J. D. **Modelagem de objetos através da UML**: the Unified Modeling Language. São Paulo: Makron Books, 1998.

[FME 2000] **Safe - User Documentation Set – Feature manipulation engine documentation**. Disponível em: <http://www.safe.com>. Acesso em:17 janeiro 2000.

[GAR 98] GARAFFA, I. M. **Análise da adequação de uma hierarquia de classes básicas para modelagem conceitual de SIG, através de um estudo de caso**. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[HAD 96] HADZILACOS, T.; TRYFONA, N. Logical data modelling for geographical applications. **International Journal of Geographical Infomation Systems,** [S.1.], v.10, n.2, 1996.

[IOC 98] IOCHPE, C.; LISBOA, F. BCH/GIS: a basic class hierarchy to support GIS conceptual design. In: INT. INTERNATIONAL CONFERENCE ON MODELING GEOGRAPHICAL AND ENVIRONMENTAL SYSTEMS GEOGRAPHICAL INFORMATIONAL SYSTEMS, 1998, Hong Kong, Cn. **Proceedings...** Hong Kong: The Chinese University of Hong Kong, 1998.

[ISO 98] ISO TERMINOLOGY PROJECT TEAM. **Collections of terms and definitions from ISO/TC211 geographic information/geomatics**. [S.1.], 1998. CD 19109.

[ISO 99] ISOWARE. **CASE-Tool REGIS**. Disponível em: <http://www.isoware.de/.1999>. Acesso em:13 maio 1999.

[KÖS 95] KÖSTERS, G.; PAGEL, B.; SIX, H. Object-oriented requirements engineering for GIS-applications. In: ACM-GIS INTERNATION WORKSHOP ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 1995, Baltimore. **Proceedings...** Baltimore: [s.n.], 1995.

[LIS 98] LISBOA FILHO, J.; IOCHPE C.; BEDARD, K. Applying analyis patterns in the GIS domain. In: ANNUAL COLLOQUIUM OF THE SPATIAL INFORMATION RESEARCH CENTRE -–SIRC, 10., 1998, Dunedin, NZ. **Proceedings...** Dunedin:SIRC, University of Otago, 1998.

[LIS 99] 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, Kansas City, USA. **Proceedings...** Kansas City: ACM Press, 1999.

[LIS 2000] LISBOA, FILHO, J. **Projeto conceitual de banco de dados geográficos através da reutilização de esquemas, utilizando padrões de análise e um framework conceitual**. Porto Alegre: PPGC da UFRGS, 2000. Tese de Doutorado.

[LIS 2000a] LISBOA FILHO, J. Modelagem de Bancos de Dados Geográficos. In: ESCOLA REGIONAL DE INFORMÁTICA DO CENTRO-OESTE, 10., 2000. **Anais...** Brasília-DF: SBC - Sociedade Brasileira de Computação, 2000. p.137-171.

[MOE 88] MOELLEWING, H. The proposed standard for digital cartographic data: report of the digital cartographic data standards task force. **The American Cartographer**, [S.1], v.15, n.1, 1988.

[NAV 92] NAVATHE, S. B. Evolution of data modeling for databases. Com**unications of the ACM**, New York, v.35, n.9, p.112-123, 1992.

[PAR 98] PARENT, C. et al. Modeling spatial data in the MADS conceptual model. In: INTERNATIONAL SYMPOSIUM ON SPATIAL DATA HANDLING, 1998, Vancouver, Canada. **Proccedings...** Vancouver: [s.n], 1998.

[RAT 99] RATIONAL SOFTWARE CORPORATION. **UML documentation – unified modeling language**. Version 1.3. Santa Clara:Rational Software Corporation, 1999. Disponível em: <http://www.rational.com/uml/index.jsp>. Acesso em: 23 março 1999.

[RAT 2000] RATIONAL SOFTWARE CORPORATION. **RationalRose2000 – Rose Extensibility User´s Guide**. Version 7.0. Santa Clara:Rational Software

Corporation, 2000. Disponível em: <http://www.rational.com/support/documentation/manuals/index.jsp>. Acesso em: 03 abril 2000.

[RAT 2000a] RATIONAL SOFTWARE CORPORATION. **Mapping object to data models with the UML**. 7.0. Santa Clara:Rational Software Corporation, 2000. Disponível em: <http://www.rational.com/products/whitepapers/101576.jsp>. Acesso em:13 maio 2000.

[RUM 91] RUMBAUCH, J.et al. **Object-oriented modeling and design**. New Jersey: Prentice-Hall, 1991.

[SAI 94] CANADA. MINISTRY OF ENVIRONMENT, LANDS AND PARKS. SURVEY AND RESOURCE MAPPING BRANCH. **SAIF/ZIP:** dataset file format, release2.0. Province of British Columbia, Canada, 1994.

[SAI 95] CANADA. MINISTRY OF ENVIRONMENT, LANDS AND PARKS. SURVEY AND RESOURCE MAPPING BRANCH. **Spatial archive and interchange format:** formal definition, release3.2. Province of British Columbia, Canada, 1995.

[SAI 96] CANADA. MINISTRY OF ENVIRONMENT, LANDS AND PARKS. SURVEY AND RESOURCE MAPPING BRANCH. **Spatial archive and interchange format:** formal functional, release1.1. Province of British Columbia, Canada,1996.

[SMI 97] SMITH, K. E.; ZDONICK, S. B. Intermedia: A Case study of the differences between relational and object-oriented database systems. In**:** CONFERENCE ON OBJECT-OPERATING PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS**,** OOPSLA, 1997, Florida. **Proceedings...** [S.1.:s.n.], 1997. p. 452-465.

[STR 98] STRAUCH, J. C. et al. **Estudo do metamodelo spatial archive and interchange format.** Rio de Janeiro: COPPE/UFRJ, 1998. Relatório Técnico

[PAR 98] PARENT, C. et al. Modelling spatial data in the MADS conceptual model. In: INTERNATIONAL SYMPOSIUM ON SPATIAL DATA HANDLING, 1998, Vancouver, Canada. **Proceedings...** Vancouver: [s.n], 1998.

HYPERLINK[PIR 93] PIRES, F. ; MEDEIROS, B.C. **Um ambiente computacional de modelagem de aplicações geográficas**. Campinas, SP: UNICAMP, 1993. Relatório Técnico.

[THO98] THOMÉ R**. Interoperabilidade em geoprocessamento:** conversão entre modelos conceituais de sistemas de informação geográfica e comparação com o padrão OPEN GIS. São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais, 1998. Tese de Mestrado.