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

Utilizando o Processo de Descoberta de Conhecimento em Banco de Dados para Identificar Candidatos a Padrão de Análise para Bancos de Dados Geográficos

por

CAROLINA MARTINS SOARES SILVA

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

Prof. Dr. Cirano Iochpe
Orientador

Prof. Dr. Paulo Martins Engel
Co-Orientador

Porto Alegre, setembro de 2003.

CIP – CATÁLOGAÇÃO DE PUBLICAÇÃO

Silva, Carolina Martins Soares

Utilizando o Processo de Descoberta de Conhecimento em Banco de Dados para Identificar Candidatos a Padrão de Análise para Bancos de Dados Geográficos/ por Carolina Martins Soares Silva. - Porto Alegre: PPGC da UFRGS, 2003.

146f.:il.

Dissertação (mestrado) - Universidade Federal do Rio Grande do Sul. Programa de Pós Graduação em Computação. Porto Alegre, BR - RS, 2003. Orientador: Iochpe, Cirano; Co-orientador: Engel, Paulo Martins.

1. Padrões de Análise. 2. Modelagem Conceitual. 3. Sistemas de Informação Geográfica. 4. Descoberta de Conhecimento em Banco de Dados. I. Iochpe, Cirano. II. Engel, Paulo Martins. III. Título.

Com saudades,
ao meu avô Antônio.

Agradecimentos

A Deus, pelo dom da vida e por permitir a conclusão de mais uma etapa.

Aos meus pais, Cecília e Paulo, pelo amor, carinho e dedicação sempre dispensados a mim e por todas as oportunidades que me propiciaram.

A minha família, avós, tios(as), primos(as) e irmão, pelo carinho e atenção dedicados a mim e pelo apoio, fundamental no cumprimento de mais essa etapa.

Ao meu orientador, Prof. Dr. Cirano Iochpe, pelos ensinamentos, discussões, oportunidades, atenção e amizade.

Ao meu co-orientador, Dr. Paulo Engel, pela atenção dispensada durante o desenvolvimento deste trabalho.

A amiga Gisa, pelos conselhos, pelos momentos felizes que compartilhamos, pelas discussões que muito contribuíram para a conclusão deste trabalho, pelo ombro amigo nos momentos difíceis, e, principalmente, pela amizade sincera.

Aos meus amigos que ficaram em Belém, que compreenderam minhas ausências e se fizeram presentes de diferentes formas enquanto morei em POA. Principalmente, Kelly, Larissa, Flávia, Fábio e Gilberto.

Aos meus novos amigos de POA, que contribuíram para tornar a jornada menos dura e abrandaram a falta que senti da minha família. Em especial a Julianita, Lucinéia, Guillermo, Luciana, Leonardo, Lito, Jane, Eliane e Roberto.

Aos colegas do MINTERCC, que compartilharam o longo período de disciplinas. Em especial à Miriam, pela companhia e amizade; ao Paulo Lima, pelo incentivo, força e por todos os momentos felizes que vivemos juntos; e ao Alécio, pela companhia, incentivo e amizade.

Aos professores Nara e Palazzo, pelas observações e direcionamentos dados durante o seminário de andamento.

Ao Ministério Público de Contas do Estado do Pará, nas pessoas do Dr. Antonio Cavalcante e Rogério Felipe, que permitiram a minha ausência para apresentação desse trabalho.

À CAPES, pelo apoio financeiro a esta pesquisa, em particular.

Ao Instituto de Informática, pela infra-estrutura disponibilizada e aos funcionários que foram solícitos e ajudaram sempre que possível.

Por fim, a todos que, de alguma forma, colaboraram para o desenvolvimento desta pesquisa.

Sumário

Lista de Abreviaturas

BD	Banco de Dados
BDG	Banco de Dados Geográficos
DCBD	Descoberta de Conhecimento e Banco de Dados
DDL	Data Description Language
DW	Data Warehouse
ER	Entidade-Relacionamento
KDD	Knowledge Discovery in Database
MD	Mineração de Dados
OO	Orientação a Objetos
SGBD	Sistema de Gerência de Banco de Dados
SIG	Sistema de Informação Geográfica
SQL	Structured Query Language
WWW	World Wide Web

Lista de Figuras

Lista de Tabelas

Resumo

Sistemas de informações geográficas (SIG) permitem a manipulação de dados espaço-temporais, sendo bastante utilizados como ferramentas de apoio à tomada de decisão. Um SIG é formado por vários módulos, dentre os quais o banco de dados geográficos (BDG), o qual é responsável pelo armazenamento dos dados.

Apesar de representar, comprovadamente, uma fase importante no projeto do SIG, a modelagem conceitual do BDG não tem recebido a devida atenção. Esse cenário deve-se principalmente ao fato de que os profissionais responsáveis pelo projeto e implementação do SIG, em geral, não possuem experiência no uso de metodologias de desenvolvimento de sistemas de informação. O alto custo de aquisição dos dados geográficos também contribui para que menor atenção seja dispensada à etapa de modelagem conceitual do BDG.

A utilização de padrões de análise tem sido proposta tanto para auxiliar no projeto conceitual de BDG quanto para permitir que profissionais com pouca experiência nessa atividade construam seus próprios esquemas.

Padrões de análise são utilizados para documentar as fases de análise de requisitos e modelagem conceitual do banco de dados, representando qualquer parte de uma especificação de requisitos que tem sua origem em um projeto e pode ser reutilizada em outro(s).

Todavia, a popularização e o uso de padrões de análise para BDG têm sido prejudicados principalmente devido à dificuldade de disponibilizar tais construções aos projetistas em geral. O processo de identificação de padrões (mineração de padrões) não é uma tarefa simples e tem sido realizada exclusivamente com base na experiência de especialistas humanos, tornando o processo lento e subjetivo.

A subjetividade prejudica a popularização e a aplicação de padrões, pois possibilita que tais construções sejam questionadas por especialistas com diferentes experiências de projeto. Dessa forma, a identificação ou o desenvolvimento de técnicas capazes de capturar a experiência de especialistas de forma menos subjetiva é um passo importante para o uso de padrões.

Com esse objetivo, este trabalho propõe a aplicação do processo de descoberta de conhecimento em banco de dados (DCBD) para inferir candidatos a padrão de análise para o projeto de BDG. Para tanto, esquemas conceituais de BDG são usados como base de conhecimento.

DCBD é o processo não trivial de descoberta de conhecimento útil a partir de uma grande quantidade de dados. Durante o desenvolvimento da pesquisa ficou claro que a aplicação do processo de DCBD pode melhorar o processo de mineração de padrões, pois possibilita a análise de um maior número de esquemas em relação ao que é realizado atualmente. Essa característica viabiliza que sejam considerados esquemas construídos por diferentes especialistas, diminuindo a subjetividade dos padrões identificados.

O processo de DCBD é composto de várias fases. Tais fases, assim como atividades específicas do problema de identificar padrões de análise, são discutidas neste trabalho.

Palavras-Chave: projeto de banco de dados geográficos, padrões de análise, descoberta de conhecimento em banco de dados, mineração de dados.

TITLE: "USING THE PROCESS OF KNOWLEDGE DISCOVERY IN DATABASES TO IDENTIFY CANDIDATES FOR ANALYSIS PATTERNS FOR GEOGRAPHIC DATABASE"

Abstract

Geographic information systems (GIS) enable users to handle spatio-temporal data. They are mainly used as decision-support tools. A GIS is composed of several functional modules, among which the geographic database (GDB), which stores data.

Even representing an important phase in the GIS project, conceptual modeling of GDB has been usually left apart. It is mainly due to the fact that people responsible for the project and implementation of GIS usually do not have experience in the use of methodologies of information systems development. The high cost of geographic data also contributes to the small attention that is given to the conceptual modeling phase of GDB design.

The use of analysis patterns has been proposed in order to aid the GDB conceptual project as well as to enable professionals with small experience to build their own schemes.

Analysis patterns are used to document database's requirements analysis and its conceptual modeling. These patterns represent any part of a requirement specification that has its origin in a project and can be re-used in other(s).

The popularization and the application of GDB analysis patterns have been harmed mainly because of the difficulty in making these constructions available to designers in general. The process of identifying new patterns (pattern mining) is not a simple task and currently has been carried out exclusively based on the experience of human experts, making the process slow and subjective.

Subjectivity harms popularization and application of patterns, because experts with different experiences can question these constructions. Thus, identification or development of techniques capable of capturing experience of experts in a less subjective way is an important step towards the use of patterns.

With this objective, the present research proposes the application of the process of knowledge discovery in databases (KDD) for the inference of candidates for analysis pattern for GDB design. In order to do that, GDB conceptual schemes are used as knowledge base.

KDD is a non-trivial process of discovering useful knowledge from a huge amount of data. In the course of research, it was clear that the application of KDD can improve pattern mining because it makes possible to examine a greater amount of schemes than what is actually carried out nowadays. This characteristic makes possible to take more schemes built by distinct human experts under consideration, which can reduce subjectivity.

KDD process is composed of several steps. These steps as well as activities, which are specific for the problem of identification of analysis patterns, are discussed in this work.

Keywords: geographic database design, analysis patterns, knowledge discovery in databases, data mining.

1 Introdução

1.1 Motivação

Sistemas de informação geográfica (SIG's) são sistemas baseados em computador que permitem a captura, armazenamento, manipulação, recuperação, análise e apresentação de dados referenciados espacialmente em relação à superfície da Terra [TEI97]. Tais ferramentas têm sido amplamente utilizadas para auxiliar no processo de tomada de decisão em áreas como, por exemplo, saúde, segurança, planejamento urbano e controle ambiental [BUR97, LIS2000, RAM94].

De modo geral, um *software* de SIG é um sistema formado por quatro grandes componentes, os quais são responsáveis pela captura, armazenamento, análise e apresentação de dados. O módulo de armazenamento, denominado banco de dados geográficos (BDG), estrutura e armazena os dados geográficos.

Dados geográficos ou geo-referenciados são comumente caracterizados a partir de suas componentes não-espacial, espacial e temporal [ARO89, PEU94]. A componente não-espacial descreve o fenômeno da realidade através de variáveis de interesse (tais como nome, extensão e vazão de um rio). A componente espacial refere-se tanto à localização espacial quanto às propriedades geométricas e topológicas da entidade ou fenômeno da realidade (coordenadas e geometria do rio, por exemplo). A data de coleta dos dados e o período de validade dos mesmos são indicados através da componente temporal.

Em virtude das características próprias dos dados geográficos, os projetos e a implementação de um BDG são mais complexos do que os de um banco de dados puramente não-espaciais. Assim como esse último, um BDG deve ser projetado, a fim de viabilizar e facilitar tanto sua compreensão, manipulação, manutenção e extensão quanto o processo de integração de dados.

O projeto de BDG ocorre em várias etapas, dentre as quais destaca-se a modelagem conceitual. Essa fase é voltada à compreensão da realidade, onde somente os elementos essenciais são enfatizados, descartando-se os elementos irrelevantes através do processo de abstração. Como resultado, as entidades do mundo real, seus relacionamentos e regras de integridade são representados através do esquema conceitual de BDG [ELM2000].

Esquemas conceituais são criados com o objetivo de documentar os tipos de dados armazenados assim como seus relacionamentos, fornecendo uma visão geral e completa das informações disponíveis para o usuário no banco de dados. A principal característica do esquema conceitual consiste na sua independência em relação ao sistema computacional a ser utilizado, possibilitando que a definição do *hardware* e do *software* de gerência do banco de dados seja postergada até a conclusão da modelagem conceitual.

A criação de esquemas conceituais é baseada em modelos de dados conceituais, os quais apresentam os construtores necessários para modelar a realidade, além das regras de utilização dos mesmos [ELM2000]. Os elementos de um esquema são, dessa forma, instâncias de uso dos construtores definidos no modelo de dados adotado.

Embora represente uma fase importante no projeto de um SIG, a modelagem conceitual de BDG tem sido, comumente, negligenciada. Esse cenário deve-se principalmente:

- à concepção do esquema conceitual não ser trivial, podendo implicar em decisões complexas, onde más escolhas conduzem a sistemas frágeis e difíceis de manter, compreender e estender [LAR2000];
- ao fato de a equipe responsável pela implantação do SIG ser, em geral, composta por profissionais que não possuem formação em informática, mas em áreas como engenharia, arquitetura, cartografia, geografia, entre outras. Esses profissionais, de modo geral, não conhecem metodologias de desenvolvimento de sistemas de informação ou são inexperientes no uso das mesmas [LIS2000]; e
- ao alto custo de aquisição dos dados geográficos, levando a que maior atenção seja dispensada a essa etapa do desenvolvimento do SIG em detrimento das demais [ARO89].

Sendo conhecidas as dificuldades em relação à modelagem conceitual de banco de dados, Coad [COA96] e Larman [LAR2000] propõem uma série de heurísticas para auxiliar no desenvolvimento dessa atividade. Paralelamente, outra abordagem que vem sendo bastante explorada na solução de problemas complexos, principalmente na área de engenharia de *software*, consiste na utilização de padrões [LIS2000, ROB93, HAY95, COA97, FOW97].

Padrão, nesse escopo, consiste em uma combinação de elementos recorrentes que apresentam a essência de uma solução para problemas análogos em contextos específicos [GAM2000, FER98]. Como são elementos de reutilização, a aplicação de padrões permite que o projeto não seja iniciado do zero, contribuindo para a redução do seu tempo total de execução. Além disso, são reutilizadas soluções consolidadas, desenvolvidas por especialistas, o que implica no aumento da qualidade dos produtos e na maior facilidade de atualização dos mesmos [JOH98].

Padrões são definidos em vários níveis de abstração, a fim de serem aplicados nas diferentes etapas de desenvolvimento de *software* e de projeto de banco de dados [BUS96]. Padrões de análise [FOW97, ROB93] são utilizados para documentar, especificamente, a análise de requisitos e a modelagem conceitual de banco de dados.

Um padrão de análise representa qualquer parte de uma especificação de requisitos que se origina em um projeto e pode ser reutilizada em outro(s). Cada padrão de análise consiste em um conjunto de classes e associações que têm algum significado no contexto de um grupo de aplicações [FER2000]. Em um nível mais baixo de abstração, um padrão de análise consiste em um subesquema conceitual de bancos de dados freqüentemente utilizado na modelagem conceitual de aplicações distintas.

Esse tipo de padrão é particularmente útil durante o projeto de BDG, pois a estrutura conceitual dos tipos de dados que compõem a chamada *cartografia básica* dos SIG é bastante semelhante para a maioria das aplicações [LIS2001]. Essa característica ainda é mais acentuada quando se considera o intercâmbio de dados espaciais, prática que vem se tornando cada vez mais comum devido ao alto custo de aquisição de tais dados.

Uma vez que a utilização de padrões tende a diminuir o tempo de projeto e aumentar a qualidade do produto final, a sua aplicação no projeto de BDG tem sido muito bem aceita pela comunidade de SIG [LIS98]. Todavia, o uso de padrões de análise no projeto de BDG tem sido prejudicado, principalmente, devido à dificuldade de disponibilização dessas construções aos projetistas em geral [JOH98]. A fim de atender a essa demanda, a criação de um catálogo de padrões de análise, específico para BDG, mostra-se de extrema utilidade.

Catálogo de padrões [APP97] é uma coleção de padrões relacionados entre si, servindo de guia de referência para a utilização dessas construções. Segundo Henninger [HEN97], a criação e utilização de catálogos voltados para o reuso deve considerar três aspectos:

- aquisição de conhecimento para sua criação;
- atualização do catálogo, de modo que ele sempre reflita as necessidades dinâmicas, inerentes às organizações e ao desenvolvimento de *software*; e
- desenvolvimento de técnicas para consultar o catálogo e recuperar os padrões de modo eficaz e eficiente.

O problema de consultar catálogos, a fim de localizar construções relevantes é, há muito, tratado como tema importante de pesquisa [BUR87 e DEV91]. Entretanto, Hennigen [HEN97] afirma que as duas outras questões não recebem a mesma atenção. Nesse contexto, o presente trabalho aborda o problema referente à aquisição de conhecimento para criação de catálogos, formados, especificamente, por padrões de análise para BDG.

A tarefa de procurar e localizar padrões a fim de documentá-los é conhecida como *pattern mining* ou mineração de padrões [ROB93, APP97, RIS99 e BUS96]. Rising [RIS99] e Bushman [BUS96] descrevem vários métodos e procedimentos utilizados com essa finalidade. No entanto, as diversas abordagens propostas são centradas na experiência prática de especialistas no domínio da aplicação, tornando o processo lento e subjetivo [FOW96, ROB93, AGE98]. Lento, pois a tarefa não é trivial e requer a análise de vários projetos previamente desenvolvidos. Subjetivo em decorrência da definição de um padrão ser baseada nos conceitos de "solução bastante testada" e "problema recorrente", cuja avaliação depende da experiência de cada indivíduo.

Padrões identificados por técnicas baseadas no conhecimento humano têm sua popularização prejudicada, pois refletem a experiência de apenas um pequeno grupo de especialistas. Essa característica permite que os padrões propostos sejam contestados por outros especialistas com diferentes experiências de projeto [FOW96]. Diante de tais dificuldades, seria promissora a utilização de uma estratégia capaz de identificar padrões de modo menos subjetivo, a qual levasse também em consideração o conhecimento de um grupo maior de indivíduos. Nesse contexto, o processo de Descoberta de Conhecimento em Banco de Dados (DCBD) apresenta-se como alternativa.

DCBD é o processo não-trivial de extração de conhecimento a partir de um grande volume de dados. Sua principal característica consiste na automatização, ao menos parcial, da tarefa de análise de dados [FAY96]. Acredita-se que o processo de DCBD possa ser utilizado para auxiliar na tarefa de mineração de padrões, pois é capaz de analisar um maior número de esquemas conceituais de diferentes especialistas em relação ao que vem sendo feito pelos métodos atualmente conhecidos. Essa característica viabiliza a identificação de padrões baseados na experiência de um número maior de especialistas em relação aos atualmente propostos. Além disso, a análise (semi)automatizada dos esquemas implica em que novos padrões podem ser identificados mais rapidamente.

Com base nessas características, esse trabalho investiga a aplicação do processo de DCBD como alternativa à solução do problema de identificação de padrões de análise para projeto de BDG, utilizando esquemas conceituais de BDG como ponto de partida.

1.2 Trabalhos relacionados

Estratégias para identificação de padrões são descritas por Rising [RIS99] e Buschman [BUS96]. Todas são, no entanto, centradas no conhecimento do especialista. Não foi possível localizar na literatura consultada qualquer esforço no sentido de automatizar a identificação de padrões de análise. Estão disponíveis diversos trabalhos voltados à criação de repositórios contendo outros tipos de construção, além de pesquisas que visam, principalmente, à identificação de instâncias de padrões de projeto. Alguns dos trabalhos consultados são descritos a seguir.

Bansiya [BAN98], Brown [BRO2001] e Shull [SHU96] apresentam alternativas para a identificação automática de instâncias de padrões de projeto, principalmente os propostos em [GAM2000]. As três abordagens baseiam-se em códigos fonte de sistemas orientados a objetos e visam identificar estruturas implementadas com base nos padrões de Gamma.

Em [BAN98] é apresentada a ferramenta DP++, a qual automatiza a detecção, identificação e classificação de padrões de projeto em programas escritos na linguagem C++. A ferramenta analisa o código fonte e, baseada nos relacionamentos estruturais entre classes e objetos, identifica instâncias de alguns padrões de projeto apresentados em [GAM2000]. O trabalho apresentado por Brown, em [BRO2001], é bastante semelhante ao citado anteriormente, sendo aplicado, todavia, para a análise de códigos escritos na linguagem Smalltalk.

Shull [SHU96] tem como principal objetivo criar bases de padrões de projeto personalizadas para cada empresa, de modo que os padrões são empiricamente validados para o ambiente em questão. Como primeiro passo nesse sentido, foi desenvolvido um método indutivo chamado BACKDOOR. O método é iterativo e, a partir de projetos já existentes, localiza tanto novos padrões quanto ocorrências de padrões já definidos. Se novas instâncias de padrões já conhecidos são localizadas nos projetos pesquisados, elas são utilizadas para atualizar métricas que avaliam a usabilidade do padrão. Por outro lado, padrões já conhecidos são empregados para deduzir características estruturais, as quais são posteriormente analisadas como potenciais padrões. Na primeira iteração, a base está vazia, sendo utilizados os padrões definidos por Gamma. O trabalho descrito em [SHU96], entretanto, apresenta apenas uma parte de todo o processo, de modo que o autor mostra somente a detecção de instâncias de uso dos padrões de projeto definidos por Gamma, não detalhando o processo de identificação de novos padrões.

Henningen [HEN97], por sua vez, busca uma alternativa para construir repositórios de componentes reutilizáveis, apresentando uma espécie de "ciclo de vida" de repositórios. O autor descreve os problemas de indexação e recuperação de componentes, além de apresentar ferramentas projetadas para suportar o refinamento incremental do repositório. No que diz respeito à população do repositório, o autor afirma que, nos estágios iniciais, ela é realizada utilizando-se, inclusive, componentes que não foram projetados com o objetivo explícito de serem reutilizados. O protótipo apresentado no trabalho extrai componentes a partir de arquivos texto de forma semi-automática, requerendo alguma intervenção do usuário. No entanto, a ênfase da pesquisa não consiste em especificar *quais* componentes serão utilizados, mas sim *como* eles são representados de modo a possibilitar a sua posterior recuperação.

Em outro contexto, Michail [MIC2000] busca identificar padrões de reuso de classes de bibliotecas específicas, a fim de facilitar a seleção e utilização de componentes disponíveis em uma biblioteca de classes.

Para atingir o objetivo, o autor aplica o algoritmo de mineração de dados de identificação de regras associativas baseado no uso de taxonomias. O algoritmo é empregado visando à identificação de classes e funções de classes da biblioteca geralmente utilizadas em conjunto em aplicações distintas.

Ainda que vários trabalhos busquem automatizar o processo de identificação de padrões ou a criação de repositórios, não foi encontrada qualquer pesquisa específica para auxílio à identificação de padrões de análise.

1.3 Objetivos

A partir da necessidade de identificar padrões de análise para BDG com base em técnicas que reduzam a influência e o esforço do especialista, o principal objetivo desse trabalho consiste em verificar a viabilidade da aplicação do processo de Descoberta de Conhecimento em Banco de Dados (DCBD) [FAY96] para esse fim. Nesse caso, a base de conhecimento utilizada é formada por esquemas conceituais de BDG.

Acredita-se que a aplicação do processo de DCBD na identificação de padrões de análise para BDG possa diminuir a subjetividade comum à seleção dos padrões, pois esse processo possibilita a obtenção de conhecimento a partir de grandes volumes de dados.

A pesquisa também busca identificar atividades específicas a serem realizadas nas várias etapas do processo de DCBD de modo a viabilizar sua aplicação no problema proposto.

1.4 Estrutura do trabalho

O texto da dissertação está estruturado em sete capítulos e três anexos, além das referências bibliográficas.

O Capítulo 2 apresenta uma introdução ao projeto de banco de dados geográficos e a padrões.

O Capítulo 3 consiste em uma breve revisão do processo de descoberta de conhecimento em banco de dados.

No Capítulo 4, é apresentada a seleção da técnica de mineração de dados utilizada no trabalho, assim como aspectos referentes à preparação dos esquemas conceituais utilizados na mineração.

O Capítulo 5 apresenta o pós-processamento das regras associativas geradas pelo algoritmo de MD. São listadas alguns tipos de regras de interesse, além de filtros utilizados com o objetivo de reduzir o número total de regras a serem analisadas pelo especialista humano.

No Capítulo 6, são descritos os testes realizados com o objetivo de verificar a adequação do processo de DCBD ao problema de identificar padrões de análise para BDG.

Finalmente, o Capítulo 7 apresenta as conclusões e possíveis trabalhos futuros.

Outras regras de interesse, além daquelas apresentadas no capítulo 5, são listadas no Anexo1. Os anexos 2 e 3 apresentam, respectivamente, detalhes dos programas de geração de dados de entrada e de pós-processamento, os quais foram utilizados nos estudos de caso realizados.

2 Projeto Conceitual de Banco de Dados Geográficos e Padrões de Análise

Sistemas de informação geográfica (SIG's) são sistemas baseados em computador que permitem a captura, armazenamento, manipulação, recuperação, análise e apresentação de dados referenciados espacialmente em relação à superfície da Terra [TEI97].

Conforme mostra a figura 2.1, um SIG é formado por vários módulos, dentre os quais o banco de dados geográficos (BDG).

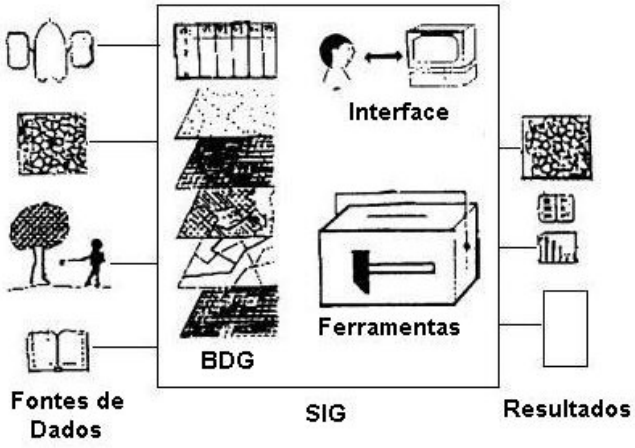


FIGURA 2.1 – Componentes de um SIG.

O BDG é o módulo do SIG responsável por estruturar e armazenar os dados de modo a possibilitar a realização das operações de análise e consulta. Nesse contexto, o projeto de banco de dados (BD) está inserido no processo de concepção do SIG como a etapa onde as estruturas estáticas do sistema são definidas, de modo a satisfazer as necessidades de usuários e aplicações.

O projeto do banco de dados é importante, pois viabiliza e facilita atividades como manutenção, extensão e integração de bancos de dados. No caso específico de BDG, o projeto mostra-se ainda mais necessário em função da prática de intercâmbio de dados geográficos, além da maior complexidade inerente aos dados manipulados por esses sistemas.

O projeto de BD é dividido nas seguintes fases ou etapas [ELM2000]:

- coleta e análise de requisitos;
- projeto conceitual;
- seleção do sistema gerenciador de banco de dados (SGBD)
- projeto lógico;
- projeto físico; e
- implementação do BD.

A primeira fase, coleta e análise de requisitos, envolve a identificação e especificação dos requisitos dos usuários. Essa etapa consiste no primeiro passo no sentido de entender os dados a serem armazenados e as regras de integridade inerentes a eles. Nesse momento, o aspecto abordado refere-se a *quais* dados devem ser armazenados e *como* eles serão processados. Essa fase não dispõe de ferramentas de

apoio formais e é, geralmente, realizada com base em entrevistas com os usuários, revisão de documentos já existentes, questionários e análise do ambiente atual.

A partir dos requisitos do usuário, o projeto ou modelagem conceitual do BD é realizado com o objetivo de produzir uma visão em alto nível do problema. O produto resultante dessa fase é o esquema conceitual do banco de dados, o qual apresenta a estrutura dos dados a serem armazenados de forma completamente independente do sistema computacional.

Na etapa de projeto lógico, o esquema conceitual criado na segunda fase é mapeado para o modelo de dados do SGBD escolhido. O resultado é o esquema lógico do BD, que, diferentemente do esquema conceitual, é dependente do SGBD.

O projeto físico do BD, por sua vez, define suas características em termos das estruturas físicas de armazenamento e caminhos de acesso. Esse é o nível mais baixo do projeto, onde as características do SGBD devem ser muito bem conhecidas.

A última etapa do projeto consiste na criação, propriamente dita, do banco de dados. Para tanto, é utilizada a linguagem de descrição de dados (*Data Description Language* -DDL) do SGBD.

Dentre todas as etapas do projeto de BD, a modelagem conceitual é considerada a mais complexa, pois os produtos das demais fases podem ser obtidos através da aplicação de regras de transformação sobre o esquema conceitual [ELM2000].

2.1 Modelagem Conceitual de BDG

Modelagem de dados é o processo de abstração através do qual apenas os elementos essenciais da realidade observada são enfatizados, descartando-se os irrelevantes. A modelagem (ou projeto) conceitual do BD compreende a descrição dos tipos de dados, além das estruturas e regras a eles aplicáveis, independente de como estes dados serão armazenados no computador. Sua importância, do ponto de vista do projetista, deve-se às seguintes características:

- torna o projeto final mais estável;
- retarda a escolha do SGBD, pois é independente do sistema escolhido;
- facilita a manutenção do BD;
- facilita a integração entre bancos de dados;
- facilita tanto a comunicação entre os projetistas quanto a comunicação entre os projetistas e o usuário; e
- aumenta a possibilidade de convergência do projeto para o produto final esperado.

Essa etapa do projeto é baseada em um modelo conceitual de dados, o qual fornece a base formal para ferramentas e técnicas que suportam a modelagem. Modelos de dados compreendem um formalismo e uma linguagem de descrição. O formalismo provê um conjunto de conceitos e regras que são utilizados no processo de modelagem da realidade. A linguagem de descrição, por sua vez, fornece a gramática e a notação para a apresentação do esquema conceitual resultante dessa etapa do projeto.

As seguintes características são requeridas em um modelo de dados:

- expressividade: o modelo deve ser capaz de diferenciar diversos tipos de dados, relacionamentos e restrições;
- simplicidade: o modelo deve ser facilmente compreendido pelo usuário final;
- minimidade: deve apresentar um número reduzido de construtores básicos, distintos entre si e semanticamente excludentes;

- representação simples: deve possuir uma representação diagramática de fácil interpretação; e
- formalidade: os conceitos do modelo devem ser definidos de forma rígida e não-ambígua.

Um formalismo bastante popular, muito empregado como modelo conceitual de bancos de dados, é o Entidade-Relacionamento (ER). Todavia, apesar de estar consolidado e possuir grande aceitação, o modelo ER não se mostra adequado para a modelagem de aplicações ditas não-convencionais, tais como BDG, bancos de dados multimídia, entre outros. Além disso, a necessidade de que o modelo represente a aplicação de maneira mais próxima a seus elementos de origem, em vez de adaptar a realidade às estruturas rígidas do computador, vem aumentando a utilização de modelos baseados no formalismo da orientação a objetos (OO) [ELM2000].

Com base no paradigma da OO, vários modelos e linguagens foram definidos para serem utilizados nas diversas etapas do ciclo de vida de *software.* O modelo que representa as estruturas estáticas e, portanto, é utilizado durante a modelagem conceitual de banco de dados, é o *diagrama de estrutura estática* ou *diagrama de classes* [LAR2000]. Apesar de apresentar vantagens em relação ao modelo ER, os modelos tradicionais da orientação a objetos também não são completamente adequados para tratar características específicas dos dados geográficos.

Dados geográficos ou geo-referenciados referem-se a entidades e fenômenos da realidade associados a sua localização referenciada espacialmente em relação à superfície da Terra [TEI97]. Tais dados são comumente caracterizados por suas componentes [ARO89, PEU94]:

- não-espacial: descreve a entidade ou fenômeno da realidade através de variáveis de interesse (por exemplo, nome do rio);
- espacial: informa a localização espacial do fenômeno, assim como suas propriedades geométricas e topológicas (as coordenadas e a geometria do rio, por exemplo); e
- temporal: indica a data de coleta e o período de validade dos dados.

Além das particularidades dos dados geográficos, outro aspecto que deve ser considerado durante a modelagem conceitual de BDG refere-se às diferentes percepções da realidade geográfica. Esta última pode ser observada segundo as visões de campo e de objeto [GOO92].

A realidade percebida na visão de campo é modelada por variáveis que possuem uma distribuição contínua no espaço como, por exemplo, pressão atmosférica, temperatura e altitude. Na visão de objetos, a realidade é constituída por entidades individuais e bem definidas, tais como edificações, vias e rios, por exemplo. As entidades percebidas segundo essa visão possuem propriedades e ocupam lugar específico no espaço, o qual não está, necessariamente, totalmente ocupado. Além disso, diferentemente do que ocorre na visão de campo, na visão de objetos, duas ou mais entidades podem estar situadas em uma mesma posição.

Em função dessas e de outras características não serem suportadas por modelos tradicionais, modelos conceituais de dados têm sido adaptados, a fim de possibilitar a modelagem de dados geográficos. Atualmente, se tem disponível uma série de modelos de dados geográficos, em sua maioria, baseados no paradigma da orientação a objetos. Dentre eles, destacam-se o padrão SAIF [BRI95], o MADS [PAR97], o UML-GeoFrame [LIS99], o GEOOA [KÖS97] e o GEO-OMT [BOR97]. Comparações detalhadas entre esses modelos estão descritas em [LIS99] e [BAS2001].

Apesar da existência de ferramentas específicas para a modelagem conceitual de BDG, essa etapa do ciclo de vida do SIG tem sido, comumente, negligenciada devido, principalmente, aos seguintes fatores:

- a concepção do esquema conceitual não é trivial, podendo implicar em decisões complexas, onde más escolhas conduzem a sistemas frágeis e difíceis de manter, compreender e estender [LAR2000];
- em geral, a equipe responsável pelo desenvolvimento do SIG é composta por profissionais que não possuem formação em informática, mas em áreas como engenharia, arquitetura, cartografia, geografia, entre outras. Esses profissionais, de modo geral, não conhecem metodologias de desenvolvimento de sistemas de informação ou são inexperientes no seu uso [LIS2000]; e
- o alto custo de aquisição dos dados geográficos acarreta que maior atenção seja dispensada a essa etapa do desenvolvimento do SIG em detrimento das demais [ARO89].

Sendo conhecidas as dificuldades em relação à modelagem conceitual de banco de dados e em virtude da importância desta fase do projeto de BDG, uma série de heurísticas foram propostas em [COA96] e [LAR2000] com objetivo de auxiliar no desenvolvimento dessa atividade. As diretrizes apontadas por esses e outros autores são bastante úteis no sentido de esclarecer dúvidas em relação a conceitos apresentados pelos modelos de dados e até mesmo na decisão de qual construtor utilizar na representação de certos aspectos da realidade. Todavia, as mesmas não garantem a criação de esquemas ótimos. Paralelamente, outra abordagem que vem sendo bastante explorada para melhorar a qualidade dos projetos, principalmente na área de engenharia de *software*, consiste na utilização de padrões [LIS2000, ROB93, HAY95, COA97, FOW97].

2.2 Padrões

Padrão, nesse contexto, é uma combinação de elementos de modelagem recorrentes em diferentes projetos e que apresentam a essência de uma solução para problemas análogos em contextos específicos [GAM2000, FER98].

A característica que torna o uso de padrões bastante interessante, tanto no contexto da engenharia de *software* quanto no projeto de banco de dados, é que essas construções são elementos de reutilização. Como tal, o emprego de padrões, além de agilizar o projeto e facilitar a comunicação entre seus integrantes, possibilita a disseminação de conhecimento e a troca de experiências entre projetistas, contribuindo para o aumento da qualidade do produto final [ROB93].

A reutilização de soluções conhecidas para resolver problemas novos há muito vem sendo realizada, de forma "não-declarada", nas várias etapas de desenvolvimento de *software*. Programadores e analistas, instintivamente, recordam experiências vividas em outros projetos e reutilizam soluções bem sucedidas. No entanto, nas duas últimas décadas, essas construções reutilizáveis tornaram-se explícitas e tangíveis [FRA95].

O pioneiro na formalização e documentação de padrões, como eles vêm sendo utilizados atualmente, foi o arquiteto Christopher Alexander, com a publicação do livro "A Pattern Language: Towns, Buildings, Construction" [ALE77].

Na ciência da computação, o processo de documentar padrões teve início em 1987, quando Ward Cunninghan e Kent Beck, especialistas em projeto de sistemas orientado a objetos, desenvolveram uma linguagem de padrões para projetar interfaces

gráficas em Smalltalk. Em 1992, foi publicado o livro de Jim Coplien [COP92] como resultado da catalogação de padrões para C++. Com a publicação da primeira edição do livro de Gamma [GAM95], o tema passou a ser mais discutido e divulgado na comunidade.

A partir da popularização do conceito, foram propostos padrões em vários níveis de abstração, a fim de serem aplicados nas diferentes etapas de desenvolvimento de *software* e de projeto de banco de dados [BUS96].

Padrões de análise [ROB93] são utilizados para documentar as etapas de análise de requisitos e modelagem conceitual. Esse tipo de padrão representa qualquer parte de uma especificação de requisitos que se origina em um projeto e pode ser reutilizada em outro(s), sendo formado por um conjunto de classes e associações que têm algum significado no contexto de uma aplicação [FER2000]. Em última instância, para o projeto conceitual de BD, padrões de análise são subesquemas conceituais freqüentemente (re)utilizados na modelagem conceitual de aplicações distintas

Segundo Fernandez [FER98, FER2000], padrões de análise prestam maior contribuição para o aumento da qualidade do *software* se comparados aos padrões de projeto, ainda que, historicamente, maior atenção seja dispensada a esses últimos. Padrões de projeto apresentam inúmeras vantagens quando aplicados na fase de projeto de aplicações. No entanto, não evitam erros na etapa de análise de requisitos, nem facilitam a construção de modelos conceituais, importantes para a compreensão da realidade. Esse fato implica em que erros cometidos em etapas anteriores ao projeto do *software* sejam propagados ainda que padrões de projeto sejam utilizados.

No caso específico de SIG, padrões de análise são particularmente úteis, pois a estrutura conceitual dos tipos de dado que compõem a cartografia básica desses sistemas é bastante semelhante para a maioria das aplicações [BUR97]. Essa característica é ainda mais valorizada se considerado o intercâmbio de dados espaciais.

Para que padrões sejam disponibilizados e utilizados pela comunidade, é necessário que estejam documentados e descritos de forma que os projetistas possam compreender sua aplicabilidade e a solução que representam. No entanto, apesar da importância de se estabelecer critérios para a descrição de padrões, ainda não existe um modelo universal para este fim. Em geral, cada autor ou grupo que propõe novos padrões opta por um modelo específico, o qual contempla os aspectos por eles julgados importantes.

Gamma [GAM2000], por exemplo, utiliza um modelo de descrição bastante extenso e rico em informações, que variam desde a motivação para aplicação do padrão até a apresentação de seu diagrama de classes de projeto.

Fowler [FOW96], Lisboa [LIS2001] e Meszaros e Doble [MES2001], por outro lado, utilizam uma forma mais simples e compacta para a descrição, a qual divide um padrão em quatro partes principais, quais sejam: seu nome; contexto onde o problema foi identificado e para o qual foi proposta a solução; problema que resolve; forças (restrições) que influenciaram na solução do problema; e solução, descrita na forma de diagrama de classes acompanhado por um texto explicativo.

Em [SHU96], os padrões são descritos através dos elementos: nome; problema; solução; e conseqüências. Sendo que esse último descreve os resultados obtidos com a utilização do padrão.

Fernandez, em [FER2001], apresenta um modelo para descrição de padrões de análise onde são especificados: o problema resolvido; as forças que influenciaram na solução; a solução propriamente dita; as conseqüências do uso do padrão; casos conhecidos onde ele foi aplicado; e padrões relacionados.

Outros modelos mais simples podem ser encontrados em [COP95] e [RIE96]. No primeiro, os padrões são descritos através da trinca problema, contexto e solução. No segundo, os autores sugerem que o par padrão-contexto, em geral, é suficiente, mas não descartam a inclusão de informações mais específicas em algumas situações.

Apesar de não existir consenso em relação às informações que devem ser utilizadas para descrever um padrão, a própria definição do termo sugere que, no mínimo, sejam descritos o problema, o contexto e a solução.

2.2.1 Catálogo de Padrões

Um catálogo de padrões [APP97] é uma coleção de padrões relacionados entre si, servindo de guia de referência para utilização dessas construções. O catálogo é uma forma de organizar e disponibilizar padrões de modo a facilitar sua aplicação.

Um catálogo de padrões geralmente subdivide essas construções em categorias e inclui referências cruzadas entre as mesmas. O catálogo impõe certa estrutura e organização a uma coleção de padrões de forma menos rígida do que as encontradas em um sistema de padrões. Esse último consiste em um conjunto coeso de padrões relacionados que trabalham em conjunto para suportar a construção e manutenção de arquiteturas completas [APP97].

O *website* de padrões [REF2000] disponível na Internet apresenta uma listagem de vários catálogos utilizados para diversos fins. Além desses, o livro "Padrões de Projeto: soluções reutilizáveis de software orientado a objetos" [GAM2000], possivelmente representa hoje o catálogo de padrões mais conhecido e utilizado pela comunidade da engenharia de *software* orientado a objetos.

Segundo Henninger [HEN97], a criação e utilização de catálogos voltados para o reuso deve considerar três aspectos:

- aquisição de conhecimento para sua criação;
- atualização do catálogo, de modo a refletir as necessidades dinâmicas inerentes às organizações e ao desenvolvimento de *software*; e
- desenvolvimento de técnicas para consultar o catálogo e recuperar os padrões de modo eficaz e eficiente.

A aquisição de conhecimento é uma atividade crítica para a criação do catálogo. Essa fase inclui a identificação dos artefatos reutilizáveis, seguida de sua compreensão e representação em um formalismo capaz de denotar suas funcionalidade e semântica. Além disso, essa etapa pode incluir generalizações, a fim de que a construção possa ser aplicada em um escopo maior [CYB2001].

A atualização do catálogo tem o objetivo de, permanentemente, refletir o conhecimento atual, já que as necessidades de usuários e aplicações mudam ao longo do tempo.

O desenvolvimento de técnicas de consulta, por sua vez, é imprescindível. Um catálogo deve conter construções suficientes para auxiliar os projetistas. No entanto, quando muitos exemplos estão disponíveis, procurar e escolher os mais apropriados se torna um problema.

Dentre os três aspectos citados, o problema de consultar catálogos, a fim de localizar construções relevantes vem sendo abordado por diversos autores [BUR87 e DEV91]. Entretanto, as duas outras questões não recebem a mesma atenção [HEN97].

A próxima seção aborda o problema da identificação de padrões, cuja solução é o objetivo deste trabalho.

2.2.2 Identificação de Padrões

A tarefa de procurar e localizar padrões a fim de documentá-los é conhecida como *pattern mining* ou mineração de padrões [ROB93, APP97, RIS99, BUS96]. Embora Shull [SHU96] afirme que não tem sido dedicado muito esforço no sentido de localizar e formalizar padrões, algumas alternativas foram encontradas na literatura.

Rising [RIS99] apresenta "técnicas" empregadas com o objetivo de identificar padrões dentro de uma organização. Dentre elas, destacam-se:

- mineração por entrevistas: consiste em entrevistar especialistas em desenvolvimento de *software* e listar as informações obtidas no formato de padrões. Os principais problemas dessa abordagem consistem em que ela é lenta e difícil de realizar. Os especialistas geralmente estão envolvidos em projetos e não possuem tempo disponível para as entrevistas. Além disso, torna-se necessário o domínio de técnicas de entrevista, a fim de que seja possível capturar todo o conhecimento que esses profissionais têm a compartilhar.
- Mineração através de compartilhamento: é uma técnica útil quando utilizada por empresas do mesmo ramo que podem compartilhar informações. Através do compartilhamento, é possível observar soluções comuns às aplicações de empresas da mesma área de atuação. O problema dessa abordagem consiste na disposição das empresas em compartilhar informações.
- Mineração em *workshops*: consiste em reunir desenvolvedores, apresentar um problema e capturar as soluções propostas. Tais soluções, sumarizadas, consistem na essência da solução do problema, ou seja, são padrões.
- Mineração da própria experiência: consiste em recordar experiências passadas na busca de pedaços de soluções para serem documentadas e compartilhadas.

Buschman [BUS96], por sua vez, lista um conjunto de atividades que podem ser realizadas no sentido de identificar novos padrões. São elas:

- Localizar, pelo menos, três exemplos onde um problema de projeto é recorrente e é resolvido de forma eficiente, usando a mesma solução. Os exemplos devem ser de aplicações reais desenvolvidas por diferentes equipes.
- Extrair a solução abstraindo detalhes específicos das aplicações. Descrever o problema resolvido e as forças a ele associadas. Listar os exemplos a partir dos quais a solução foi originada.
- Declarar a solução como sendo um candidato a padrão.
- Promover um *workshop* para melhorar a descrição do candidato a padrão e compartilhá-lo com outros projetistas.
- Aplicar o candidato a padrão em projetos reais.
- Promover o candidato a padrão, se a aplicação for bem sucedida. Caso contrário, melhorar sua descrição e testá-lo novamente ou descartá-lo.

Uma característica destacada por Fowler [FOW96], que pode ser observada tanto no trabalho de Rising quanto no de Buschman, consiste na forte dependência dos padrões em relação à experiência de especialistas. Por ser fortemente dependente do conhecimento do ser humano e ser realizado de modo manual, o processo de identificação de padrões tem se mostrado lento e subjetivo [ROB93, AGE98].

O processo é lento, porque requer do analista habilidade, experiência e capacidade de abstração, além de conhecimento de vários projetos anteriores [LIS2001,

ROB93]. O processo é subjetivo em decorrência da definição de padrão, baseada nos conceitos de "solução bastante testada" e "problema recorrente", os quais estão diretamente relacionados com as experiências de projeto de cada especialista [AGE98]. Tais características prejudicam a popularização e a aplicação de padrões, contribuindo para a escassez de soluções reutilizáveis. O caráter subjetivo é particularmente prejudicial, pois possibilita que os padrões propostos por um especialista (ou grupo de especialistas) sejam contestados por outros com diferentes experiências de projeto [FOW96]. Esse cenário é verificado inclusive nas áreas de engenharia de *software* e projeto de BD, as quais apresentam uma enorme quantidade de informação (códigos fonte, esquemas de banco de dados) a partir da qual é possível obter componentes reutilizáveis.

Em contraste à escassez de padrões, Kerth, em [FRA95], afirma que, se cada especialista apontar pelo menos uma construção a qual julga útil, correta e recorrente, o número de padrões crescerá tão rapidamente que se tornará impossível saber quais existem e quais problemas cada um deles soluciona.

Se por um lado, a maneira como o processo de identificação de padrões vem sendo conduzido não se mostra eficiente, por outro não se tem visto muito esforço no sentido de desenvolver métodos alternativos capazes de realizar essa atividade de forma automática e menos dependente do conhecimento do especialista humano [SHU96, HEN97].

Diante desse cenário, o presente trabalho tem como objetivo verificar a viabilidade de utilização do processo de DCBD para identificar padrões de análise para BDG que levem em consideração o conhecimento de um número maior de especialistas em relação aos métodos atualmente empregados. O capítulo seguinte apresenta uma breve descrição do processo de DCBD.

3 Descoberta de Conhecimento em Banco de Dados

A crescente massa de dados armazenada pelas organizações e o grande número de variáveis a serem consideradas durante as análises de dados consistem na principal motivação para o desenvolvimento e a aplicação de técnicas de Descoberta de Conhecimento em Banco de Dados (DCBD) [PIA91, FAY96].

DCBD é o processo não-trivial de extração de conhecimento a partir de um grande volume de dados. O processo é considerado não-trivial, pois o conhecimento está representado de forma implícita, não sendo possível obtê-lo diretamente, a partir de consultas convencionais em SQL, por exemplo. Assim, DCBD busca desenvolver métodos e técnicas capazes de obter conhecimento a partir de dados brutos, automatizando, ao menos parcialmente, a tarefa de análise dos dados.

Fayyad [FAY96] apresenta o problema da descoberta de conhecimento como o mapeamento de dados em baixo nível (registros de BD, por exemplo) para outra forma mais compacta, abstrata ou útil (como, por exemplo, modelos ou regras).

O processo de DCBD é interativo e iterativo. Para ser empregado, requer a perfeita compreensão do domínio da aplicação e dos objetivos a serem alcançados.

O processo pode ser dividido nas fases de *preparação de dados, mineração de dados* e *pós-processamento*. Cada fase é formada por etapas, as quais incluem a realização de várias atividades, voltadas a atingir objetivos específicos. A figura 3.1 mostra a decomposição desse processo.

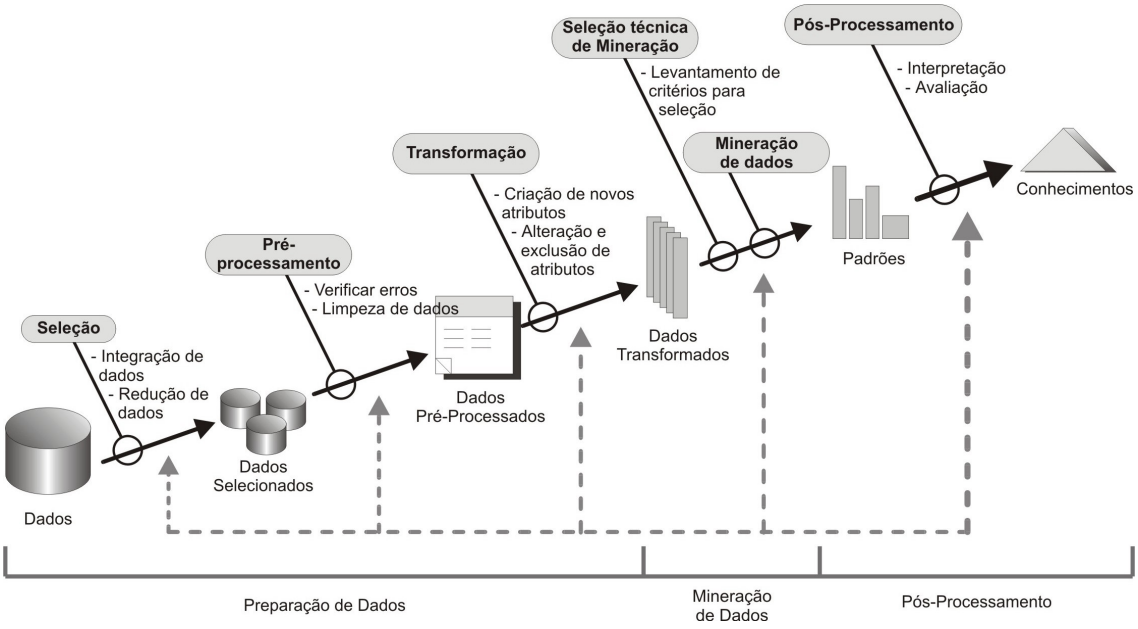


FIGURA 3.1 – Fases, etapas e atividades inerentes ao processo de DCDB.

Alguns autores tratam a mineração de dados (MD) como sinônimo de descoberta de conhecimento em banco de dados, enquanto existe uma corrente que diferencia os dois conceitos e define a mineração de dados como uma etapa do processo completo de DCBD [FAY96, ADR97, NAV2000]. No contexto desse trabalho, será utilizada essa última abordagem.

As seções a seguir descrevem cada uma das fases do processo de DCBD, assim como suas etapas e algumas das atividades que podem ser realizadas. Maior ênfase é dispensada à MD, pois, além de ser um dos focos principais do trabalho, essa fase pode

ser estudada de forma independente da aplicação. A fase de pós-processamento e algumas das etapas da preparação de dados também foram estudadas em detalhes. Todavia, como nesses últimos casos as atividades e decisões estão diretamente relacionadas à aplicação, tais etapas são detalhadas nos capítulos 4 e 5.

3.1 Preparação de Dados

Na fase de preparação de dados são realizadas atividades que visam tanto a aumentar a qualidade dos dados quanto a dispor os mesmos de modo que a extração de conhecimento seja facilitada. A principal contribuição dessa fase deve ser o aumento da acurácia e da eficiência na etapa de mineração de dados [HAN2001].

A preparação de dados é uma fase muito importante para o sucesso da aplicação de DCBD, pois a qualidade das decisões está diretamente relacionada aos dados nos quais elas se baseiam e os bancos de dados disponíveis estão, geralmente, incompletos, com erros e inconsistentes [HAN2001]. Além disso, uma vez que o projeto do banco de dados não é realizado visando à aplicação do processo de DCBD, algumas de suas características dificultam a mineração de dados.

A fase de preparação de dados é composta das etapas de seleção, pré-processamento, limpeza e transformação de dados. Cada uma dessas etapas envolve a aplicação de uma ou várias técnicas que visam preparar melhor os dados para serem minerados.

Na etapa de seleção de dados, busca-se criar a base a ser utilizada durante o processo, através da identificação dos dados relevantes para atingir os objetivos do usuário. Não é raro que sejam utilizadas mais de uma fonte de dados. Nesse caso, torna-se necessário que os dados das diferentes fontes sejam integrados em um único repositório, a fim de serem combinados de modo consistente. A integração de dados, em geral, não é uma atividade simples e visa resolver uma série de problemas como redundância e ocorrência de sinônimos, homônimos e parônimos.

Outro aspecto considerado refere-se à quantidade de dados. Uma vez que a análise de enormes volumes de dados pode ser impraticável, técnicas de *data reduction* são utilizadas com o objetivo de reduzir o volume de dados, obtendo uma representação que mantém a integridade dos dados originais [HAN2001].

A etapa de pré-processamento visa identificar erros nos dados selecionados, localizando problemas como atributos com nomes errados, atributos obrigatórios sem valor, registros contendo uma mesma informação codificada de forma diferente e/ou registros duplicados.

Após a identificação dos erros, é realizada a "limpeza" dos dados, a fim de resolver os problemas identificados. Para tanto, são fundamentais a ajuda de um especialista no domínio da aplicação e a compreensão do uso que será feito do conhecimento adquirido.

Durante a etapa de transformação, os dados são modificados em função dos objetivos da aplicação do processo de DCBD e do formato requerido pela técnica de mineração de dados a ser utilizada. Essa etapa é necessária, pois, em geral, as bases de dados não são criadas com objetivo de serem utilizadas para descoberta de conhecimento, o que as torna, em geral, inadequadas para o processo de DCBD.

3.2 Mineração de Dados (MD)

Essa fase do processo é dividida em duas etapas. A primeira visa à avaliação das técnicas de mineração de dados, a fim de verificar qual é a mais adequada para os

objetivos da utilização do processo de DCBD. Depois de selecionada a técnica, passa-se à etapa também denominada de mineração de dados, na qual algoritmos que implementam a técnica escolhida são aplicados sobre os dados pré-processados para identificar padrões interessantes existentes na base. Nesse contexto, padrão é a expressão **E** que descreve ou apresenta um subconjunto $\mathbf{F_E}$ de **F**, formado por fatos. **E** caracteriza-se por ser mais simples que a enumeração de todos os fatos em $\mathbf{F_E}$.

A etapa de mineração de dados consiste na utilização de algoritmos de análise para identificar regras ou modelos estatísticos os quais representam conhecimento implícito em uma grande quantidade de dados [FAY96]. Esse processo, geralmente, envolve aplicações interativas e iterativas de um método particular com o objetivo de produzir conhecimento novo de interesse do usuário. Essa etapa do processo de DCBD visa única e exclusivamente encontrar padrões, cabendo às demais etapas garantir a qualidade e a utilidade do conhecimento adquirido.

O uso de técnicas de MD com o objetivo de substituir o trabalho do especialista traz como benefícios [BRA98]:

- geração de modelos com menor esforço manual, tornando o processo mais eficiente;
- a possibilidade de geração e avaliação de um número maior de modelos, aumentando a probabilidade de encontrar o melhor deles; e
- pode ser utilizada por analistas menos experientes, uma vez que algumas atividades são automatizadas.

Os objetivos de mais alto nível da mineração de dados são predição e descrição [FAY96]. A predição baseia-se em um conjunto de variáveis conhecidas, as quais são usadas para prever valores futuros de outras variáveis de interesse, gerando modelos do tipo preditivo. A descrição, por outro lado, visa encontrar padrões que descrevam os dados e sejam interpretáveis pelo ser humano, através da criação de modelos descritivos.

Para gerar os modelos, várias técnicas estão disponíveis, cada uma mais adequada à realização de uma tarefa específica. Dentre as tarefas realizadas pela MD, destacam-se classificação, regressão, agrupamento e análise de associações, as quais são descritas na seção 3.2.1.3.

No que se refere à etapa de seleção da técnica de mineração de dados, alguns fatores devem ser levados em consideração, dentre os quais destaca-se o objetivo do usuário. Para Berry e Linnoff [BER97], um aspecto determinante na escolha da técnica é a identificação da tarefa a ser realizada, sendo de fundamental importância que a técnica seja compatível e aplicável à sua realização. Se o objetivo do usuário é prever o comportamento de seus clientes, por exemplo, não será útil aplicar técnicas cujos modelos gerados simplesmente descrevam os dados. Outros fatores como o tipo de repositório a ser minerado e o formato de representação do conhecimento adquirido também devem ser levados em consideração durante a seleção da técnica de MD [GOE99].

Na fase de mineração de dados, a etapa que demanda maior esforço consiste na seleção da técnica, uma vez que após essa escolha e a preparação dos dados, sua aplicação é trivial.

3.2.1 Critérios para seleção de Técnica de MD

Vários fatores são levados em consideração durante a escolha da técnica de MD, destacando-se:

- tipo de modelo gerado;

- tipo de aprendizado (supervisionado/não-supervisionado; em lote/incremental);
- tarefa a ser realizada;
- tipo de repositório a ser minerado; e
- formato de representação do conhecimento.

As próximas seções apresentam cada um dos aspectos acima relacionados, assim como a descrição de algumas técnicas estudadas com base nesses elementos.

3.2.1.1 Tipo de Modelo

Conforme citado anteriormente, a mineração de dados é aplicada para gerar regras ou modelos estatísticos os quais representam conhecimento implícito em uma grande quantidade de dados. Os dois principais tipos de modelos gerados são o preditivo e o descritivo.

Um modelo preditivo é gerado a partir de um conjunto de variáveis conhecidas, utilizadas para prever valores futuros de outras variáveis de interesse. Para obter um modelo preditivo é necessária a existência de um banco de dados contendo vários exemplos que disponibilizem, inclusive, o valor já apresentado no passado das variáveis a serem previstas. De posse de um modelo desse tipo é possível responder a perguntas como "Es sa transação é fraudulenta?" ou "Quanto esse cliente irá produzir de lucro?".

Modelos do tipo descritivo, por sua vez, descrevem os dados e devem ser interpretáveis pelo ser humano. Esse tipo de modelo disponibiliza informações a respeito do relacionamento entre os dados como, por exemplo, "peso e idade, em conjunto, são os principais fatores de ocorrência da doença x".

Um modelo puramente preditivo busca a acurácia, enquanto um puramente descritivo tem seu foco na compreensão da realidade.

3.2.1.2 Tipo de Aprendizado

A fase em que o modelo é construído é comumente chamada de aprendizado, podendo ser classificado segundo o grau de intervenção humana e quanto à forma de manipulação dos dados de entrada. De acordo com o primeiro critério, grau de intervenção humana, o aprendizado é dito supervisionado ou não supervisionado.

Aprendizado supervisionado é baseado em exemplos e, geralmente, é utilizado por técnicas que realizam predição. Nesse caso, os dados de entrada são previamente classificados por um especialista no domínio da aplicação e são utilizados como exemplos pelo algoritmo de MD. O algoritmo "analisa" todos os exemplos de cada classe informada pelo especialista e procura um modelo que descreva esses dados de modo a classificar corretamente o maior número de casos possível.

O aprendizado não-supervisionado, por sua vez, é utilizado para descrição e busca identificar como os dados estão relacionados, quais itens são similares, quais são diferentes e de que forma. Algoritmos cujo aprendizado é do tipo não-supervisionado agrupam padrões apresentados na entrada de acordo com a similaridade existente entre eles. Nesse caso, nenhuma classe é predefinida, cabendo ao próprio algoritmo manipular os dados de modo a determiná-las.

No que se refere à estratégia de manipulação dos dados de entrada, o aprendizado pode ser em lote ou incremental. Algoritmos de aprendizado em lote consideram todo o conjunto de dados de uma única vez. A modificação do arquivo de entrada implica na necessidade de execução de todo o processo de DCBD novamente. Algoritmos

baseados em aprendizado incremental não desconsideram resultados de minerações anteriores quando o arquivo de entrada é modificado.

3.2.1.3 Tarefas realizadas pela MD

Para que a aplicação do processo de DCBD seja satisfatória e leve aos objetivos esperados, é imprescindível que o problema e o domínio da aplicação sejam muito bem conhecidos. Apenas após a compreensão do problema, é possível verificar qual técnica de MD se aplica na sua solução. Por isso, a identificação da tarefa a ser realizada é o principal aspecto considerado durante a escolha da técnica de MD [BER97].

Abaixo são descritas algumas das principais tarefas nas quais a MD se aplica. Outros exemplos são encontrados em [CHE96] e [FAY96].

a) Classificação

Consiste em aprender uma função que mapeia (classifica) um item para uma dentre várias classes pré-definidas. É uma das aplicações mais comuns da mineração de dados [BRA98a], sendo utilizada nas áreas de *marketing*, mercado financeiro, telecomunicações, seguros, medicina, entre outras.

Técnicas de classificação são baseadas em aprendizado supervisionado e geram modelos do tipo preditivo [BRA98a].

A acurácia do modelo é verificada na fase de testes, a partir do cálculo do percentual de casos do conjunto de dados de teste corretamente classificados pelo modelo gerado. Durante esse processo, o resultado também é verificado a fim de detectar se o mesmo não está "condicionado", ou seja, específico para os dados usados no treinamento.

Exemplo típico dessa tarefa é a classificação de clientes que buscam concessão de crédito em baixo, médio ou alto risco.

b) Regressão

É muito semelhante à classificação, sendo as duas diferenciadas pelo tipo de dado retornado como saída. No caso da regressão, não são obtidos valores discretos, representando rótulos de classe, mas sim valores contínuos. Por este motivo, técnicas desse tipo são utilizadas quando a saída da previsão encontra-se em um intervalo de valores e não é uma classe categórica.

Técnicas de regressão são utilizadas com vários objetivos, tais como estimar a probabilidade de sobrevivência de um paciente com base em testes de diagnóstico, prever a demanda por um novo produto como uma função dos gastos com propaganda ou prever a quantidade de biomassa existente em uma floresta a partir de medidas de microondas.

O arquivo de treinamento utilizado na regressão é semelhante ao da classificação. No entanto, os exemplos estão associados a um valor numérico específico conhecido previamente. O tipo de aprendizado é supervisionado e o modelo gerado é preditivo.

Assim como em algumas outras tarefas, o valor da predição é geralmente menos importante do que a estrutura da descrição aprendida, expressa em termos de como os atributos importantes estão relacionados para atingir o valor da saída [WIT2000].

c) Agrupamento

Técnicas de agrupamento geram modelos descritivos que buscam agrupar objetos em classes de objetos similares.

Os dados de entrada são analisados de modo a identificar classes naturais (ou grupos), onde elementos pertencentes a uma mesma classe possuem grande similaridade entre si (similaridade intra-grupo) e elementos de classes diferentes são pouco semelhantes (similaridade inter-grupos). Cabe ao usuário determinar o significado de cada classe encontrada.

Técnicas que realizam essa tarefa normalmente não são baseadas em exemplos previamente rotulados, sendo caracterizadas, portanto, por utilizarem aprendizado do tipo não-supervisionado. O sucesso do agrupamento é medido subjetivamente com base no quanto o resultado é útil ao usuário humano [WIT2000].

Exemplos de aplicação de agrupamento são a identificação de diferentes grupos de clientes de uma loja, a categorização de genes com funcionalidades similares ou a identificação de áreas cujo uso do solo é comum.

Segundo Witten e Frank [WIT2000], o resultado dos algoritmos de agrupamento é um diagrama que mostra a distribuição das instâncias pelos grupos. Berry e Linnoff [BER97] afirmam que a tarefa de agrupamento geralmente é realizada antes da utilização de outra técnica de MD ou modelagem, o que torna o agrupamento apenas um passo no sentido de descrever os dados.

Brand e Gerritsen [BRA98] afirmam que técnicas que realizam agrupamento são muito subjetivas, devido à necessidade de se definir medidas de distância.

d) Análise de Associações

Busca encontrar relacionamentos interessantes entre itens de um grande conjunto de dados de um domínio específico. Os relacionamentos identificados representam padrões de comportamento e são descritos através de regras associativas (ou regras de associação).

Técnicas que realizam esse tipo de tarefa são baseadas em aprendizado não-supervisionado e são utilizadas tanto para descrição quanto para predição.

Esse tipo de análise é muito utilizado no problema da cesta de compras (*market basket analysis*), cujo objetivo é verificar os hábitos de consumo dos clientes de uma loja, através da verificação de quais grupos de produtos são normalmente adquiridos em uma visita ao estabelecimento. Apesar de essa ser a aplicação mais famosa, essa tarefa também é utilizada em aplicações médicas para verificar, por exemplo, que a ocorrência de uma dada enfermidade está associada à ocorrência de outra.

3.2.1.4 Tipos de Repositório a ser Minerado

Em tese, a mineração de dados pode ser aplicada sobre qualquer tipo de repositório de dados, mas os algoritmos e dificuldades de implementação, em geral, diferem de acordo com o tipo de repositório [HAN2001]. Os principais tipos de repositório utilizados para mineração são os bancos de dados relacionais, *data warehouses (DW),* bancos de dados transacionais, bancos de dados orientados a objetos, bancos de dados objeto-relacionais, bancos de dados geográficos, banco de dados temporais, banco de dados de texto, bancos de dados multimídia e a *World Wide Web (WWW).*

Os bancos de dados relacionais e transacionais e os *data warehouses* são os repositórios mais freqüentemente utilizados para MD. A aplicação de MD sobre os

demais tipos de repositório citados é considerada desafio na área [FAY96a, CHE96, HAN2001].

A mineração de *data warehouses* é facilitada, pois durante a criação do DW algumas das atividades comuns à etapa de preparação de dados da DCBD já são realizadas.

A MD em bancos de dados orientado a objetos e em bancos de dados objeto-relacionais é muito semelhante [HAN2001]. No entanto, se comparada à mineração realizada sobre bancos de dados relacionais, novas técnicas devem ser desenvolvidas, a fim de lidar com os novos conceitos inerentes a esses repositórios.

Em bancos de dados geográficos podem ser aplicados algoritmos de MD com o objetivo de identificar padrões que levam em consideração a referência espacial das informações. Por exemplo, descrição das características de casas localizadas próximo a parques ou praças.

Técnicas de MD são aplicadas sobre os dados armazenados em bancos de dados temporais, a fim de encontrar características referentes à evolução dos objetos ao longo do tempo ou padrões de mudanças de objetos.

No que se refere à mineração de dados na WWW, essa aplicação consiste em uma das mais pesquisadas atualmente, em virtude da grande quantidade de dados disponível na *web* e na crença de que existe uma enorme quantidade de conhecimento disperso pela rede.

3.2.1.5 Formato de Representação do Conhecimento

Os padrões identificados por técnicas de MD podem ser representados em diversos formatos, os quais são classificados em caixa preta ou caixa transparente [WIT2000]. Os dois tipos de formato podem fazer boas predições, diferindo entre si pela forma de representação dos padrões. Formatos do tipo caixa transparente representam os padrões em termos de uma estrutura que pode ser examinada, sobre a qual é possível raciocinar e que pode ser usada para guiar decisões futuras. Os do tipo caixa preta, por sua vez, não apresentam a estrutura do padrão de modo explícito. Independente do tipo, o formato de representação dos padrões está diretamente relacionado com a técnica utilizada.

A seguir são apresentados alguns formatos de representação de conhecimento, comumente utilizados.

a) Tabela de Decisão

É a forma mais rudimentar de representar o que foi aprendido pela máquina. A sua interpretação é simples, mas a criação envolve a seleção dos atributos que devem ser apresentados. Quanto menor o número de atributos, mais simples e legível é a tabela. No entanto, essa escolha não é simples e consiste no ponto crítico da criação da mesma.

A tabela 3.1 é um exemplo de uma tabela de decisão. Sua interpretação é bastante direta: para verificar se a atividade "jogar" é realizada, é necessário apenas analisar as condições determinadas pelos demais atributos.

TABELA 3.1 – Exemplo de tabela de decisão

Tempo	Umidade	Presença de vento	Joga
Ensolarado	Alta	Sim	Não
Ensolarado	Baixa	Não	Sim
Encoberto	Alta	Não	Sim
Encoberto	Baixa	Sim	Sim
Chuvoso	Alta	Não	Sim
Chuvoso	Baixa	Não	Sim
Chuvoso	Baixa	Sim	Não

b) Árvore de Decisão

É uma estrutura de decisão baseada em árvore onde os nós correspondem a atributos a serem testados e as folhas são as classes do problema. Os arcos apresentam os valores que os atributos podem assumir.

Em geral, cada teste em um nó compara um valor de atributo com uma constante. Cada folha corresponde à classe de todas as instâncias que a atingem. A classificação de novas instâncias se dá percorrendo a árvore de cima para baixo, de acordo com os valores dos atributos testados, até atingir uma folha.

A figura 3.2 apresenta um exemplo de árvore de decisão, representando as informações da tabela 3.1.

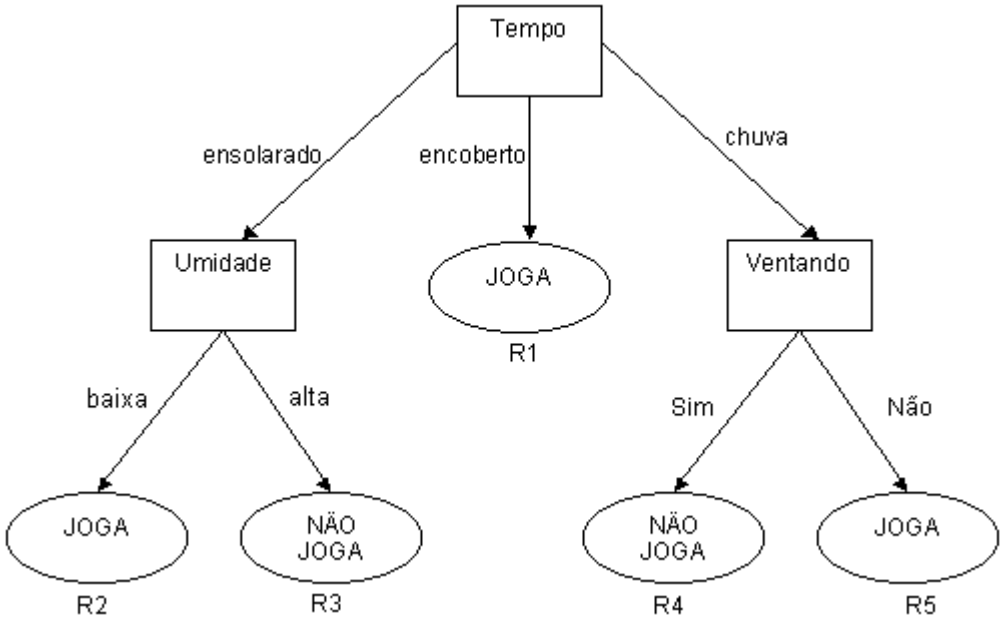


FIGURA 3.2 – Exemplo de árvore de decisão

c) Regras de Classificação

São uma alternativa às árvores de decisão. Sua estrutura é A → C (A implica C), onde A é o antecedente e C, o conseqüente da regra. O antecedente consiste em uma série de testes e o conseqüente, indica a classe ou classes na(s) qual(is) as instâncias cobertas pela regra se enquadram. Em geral, as pré-condições apresentadas no antecedente da regra devem ser todas satisfeitas para que a mesma se aplique.

A geração de regras de classificação a partir de árvores de decisão é direta. Para cada folha da árvore é gerada uma regra. O conseqüente da regra é a classe designada

pela folha, enquanto seu antecedente inclui as condições (ligadas pelo conectivo lógico "e") de todos os nós entre a raiz da árvore e essa folha específica.

Exemplos de regras de classificação geradas a partir da árvore de decisão da figura 3.2 são:

R1: tempo = encoberto → JOGA
R2: tempo = ensolarado e umidade = baixa → JOGA
R3: tempo = ensolarado e umidade = alta → NÃO JOGA
R4: tempo = chuvoso e ventando = sim → NÃO JOGA
R5: tempo = chuvoso e ventando = não → JOGA

d) Regras Associativas

Regras associativas são formadas por condições atributo-valor que ocorrem simultaneamente com bastante freqüência. São muito semelhantes às regras de classificação. No entanto, predizem o valor de qualquer atributo e não apenas o que corresponde à classe [WIT2000]. Isto é, os itens do conseqüente das regras associativas correspondem a qualquer atributo da tabela. A mineração de regras associativas a partir da tabela 3.1 poderia identificar, por exemplo, as seguintes regras: *tempo = encoberto → JOGA* e *JOGA → umidade = baixa*. A mineração de regras de classificação não permite identificar, em um único processamento, as duas regras anteriormente apresentadas. Para alterar a classe, torna-se necessário realizar a MD novamente.

A flexibilidade das regras associativas em relação às regras de classificação implica em que, para o mesmo conjunto de dados, seja gerado um número muito maior de regras associativas do que de regras de classificação. Sendo assim, para limitar a quantidade de regras encontradas, são consideradas apenas aquelas aplicáveis sobre um número significativo de instâncias e que possuem grande acurácia. Medidas objetivas são definidas para avaliar essas características. São elas, o suporte e a confiança.

O suporte de uma regra corresponde ao percentual de transações apresentadas na entrada que contém todos os itens apresentados tanto no conseqüente quanto no antecedente da mesma (A U C). A confiança da regra é a relação entre o número de transações que contém todos os seus itens (A U C) e o número de transações que contém, pelo menos, os itens do seu antecedente (A). Apenas são apresentadas no resultado regras com valores de suporte e confiança maiores do que os mínimos especificados pelo usuário.

Regras associativas que apresentam mais de um atributo no conseqüente devem ser interpretadas com cuidado. A regra "vento = não e joga = não → tempo = ensolarado e umidade = alta" não é resultante da combinação das regras: "vento = não e joga = não → tempo = ensolarado" e "vento = não e joga = não → umidade = alta". O suporte das três regras é o mesmo, pois é calculado com base em todos os atributos. No entanto, não se pode afirmar que a confiança das duas últimas regras apresentadas é a mesma da primeira, pois tanto o conseqüente quanto o antecedente das regras são diferentes entre si.

Com base na primeira regra, é possível afirmar que a regra "vento = não e joga = não e umidade = alta → tempo = ensolarado" também se aplica e o valor de sua confiança é, com certeza, igual ou superior ao da primeira.

e) Centróide

O centróide geralmente é criado por técnicas de agrupamento, podendo ser alterado e refinado durante a formação dos grupos ou na inclusão de novos elementos.

O centróide é uma média das características dos elementos do grupo, utilizado para representar todos os objetos do grupo. Por esse motivo, ele não é capaz de representar fielmente todas as características do grupo, nem cada elemento individualmente [WIL88].

Em geral, ele é representado por um vetor de valores.

3.2.2 Técnicas de Mineração de Dados

Várias são as técnicas de MD disponíveis para serem aplicadas nas diversas tarefas anteriormente citadas. Algumas das técnicas estudadas no decorrer deste trabalho são discutidas a seguir.

Técnicas de indução de árvore de decisão são utilizadas para construir árvores de decisão a partir de uma base de dados de treinamento. O processo de construção da árvore é guiado heuristicamente pela escolha do atributo mais significativo a cada passo. A seleção do atributo busca sempre minimizar a profundidade da árvore e, conseqüentemente, a quantidade de testes necessários para obter o resultado da classificação. O método de construção da árvore e o resultado final dependem diretamente da escolha do algoritmo de indução.

Vários algoritmos estão disponíveis para a indução de árvores de decisão. Um dos primeiros e mais conhecidos é o ID3 [QUI86]. Após ele, surgiram ainda o CART [BRE84], o CHAID [KAS80] e o C4.5 [QUI93]. A aplicação desses algoritmos gera modelos do tipo preditivo.

Apesar de também realizar análise de associações, técnicas de indução de árvores de decisão são tipicamente utilizadas para classificação. O conhecimento adquirido pela aplicação da técnica é representado em forma de árvore de decisão, a partir da qual são geradas regras de classificação.

A técnica de identificação de regras associativas, por sua vez, tem como objetivo descrever relacionamentos freqüentes entre os dados. O arquivo de entrada do algoritmo consiste em um conjunto de transações contendo, cada uma, uma relação de itens. A partir desses dados, busca-se identificar a relação de dependência existente entre um item (ou conjunto de itens) e outro.

Técnicas de identificação de regras associativas geram modelos descritivos e preditivos. O resultado da aplicação da técnica consiste em um conjunto de regras associativas, as quais apresentam a maneira pela qual os dados estão relacionados. Outra característica dessa técnica consiste em que ela utiliza aprendizado do tipo não-supervisionado.

O outro tipo de técnica estudada, as redes neurais artificiais (RNA), ou simplesmente redes neurais (RN), são resultado de uma metáfora computacional para o funcionamento do cérebro humano. Elas surgiram em 1943, no trabalho de McCulloch e Pitts [MCC43], que foi a primeira tentativa de simular, no computador, o comportamento do neurônio biológico.

Outros modelos surgiram posteriormente, no entanto, por serem limitados no que diz respeito à capacidade de aprendizado e só conseguirem resolver problemas linearmente separáveis, as pesquisas na área foram desaceleradas. Um novo impulso só foi dado em meados da década de 80, quando foi apresentada a solução para as limitações detectadas.

A partir de então, as RNA tornaram-se uma tecnologia largamente utilizada para resolver problemas tanto de classificação quanto de agrupamento, de modo que uma série de arquiteturas foram propostas para esses fins. Nesse contexto, foram estudados o algoritmo *backpropagation* e o Modelo Neural Combinatório (CNM – *Combinatorial Neural Model*).

O algoritmo *backpropagation* é utilizado para atualizar os pesos da rede

Multilayer Perceptron (MLP), bastante aplicada na tarefa de classificação.

O *backpropagation* utiliza vetores de exemplo associados a rótulos de classe para modificar os pesos. Cada um dos vetores de exemplo é apresentado na entrada da rede e a saída resultante é comparada com o rótulo de classe associado ao vetor. A diferença entre o valor encontrado pela rede e a saída desejada é utilizada para atualizar os pesos. O conhecimento da rede está distribuído pelos pesos, por isso diz-se que a representação é do tipo caixa preta.

O modelo neural combinatório [MAC89] é uma rede neural que utiliza o conceito de predicado, a fim de tratar informações categóricas. O CNM possui uma arquitetura conectada adiante com três ou mais camadas e visa encontrar regras de produção que associem evidências de entrada a hipóteses de saída.

Os neurônios da camada de entrada correspondem aos predicados e representam as evidências do domínio da aplicação. As camadas intermediárias são chamadas combinatórias porque representam as diferentes combinações dos dados de entrada. A camada de saída contém neurônios os quais representam as diferentes hipóteses (classes) do domínio do problema.

O CNM gera modelos do tipo preditivo e pode ser aplicado tanto para classificação quanto para análise de associações. Assim como outras técnicas que realizam classificação, o aprendizado é do tipo supervisionado e o conhecimento é representado através de regras de classificação.

A técnica näive-bayes tem por objetivo identificar relacionamentos entre variáveis, a fim de gerar um modelo de classificação. A técnica baseia-se no teorema de Bayes para calcular as probabilidades de uma nova ocorrência pertencer a cada uma das classes.

A aplicação dessa técnica implica em que todas as variáveis utilizadas sejam estatisticamente independentes entre si, o que nem sempre é real. No entanto, segundo [BRA98a], mesmo com esse problema, a técnica produz bons resultados na identificação de relacionamentos simples.

A näive-bayes caracteriza-se por gerar modelos preditivos, voltados, principalmente, para a classificação. O resultado gerado é representado por regras de classificação e o tipo de aprendizado da técnica é supervisionado.

K-médias é uma das técnicas de *clustering* mais conhecidas, sendo empregada para identificar agrupamentos em dados do tipo numérico (valores contínuos). Essa técnica é baseada em centróide, o qual representa o valor médio dos objetos de cada grupo.

A técnica é implementada de modo que primeiramente é realizada a seleção aleatória de *k* objetos do arquivo de entrada, cada um representando o centro de um agrupamento. Para cada grupo, são designados os objetos mais similares com base na distância entre os objetos e o centro do grupo. Em seguida, uma nova média é computada para cada grupo e o processo é executado recursivamente.

O algoritmo de agrupamento demográfico também é utilizado para encontrar agrupamentos. No entanto, diferentemente do k-média, ele é aplicado sobre dados categóricos. Os agrupamentos são criados a partir da comparação de cada objeto com todos os agrupamentos criados. O algoritmo utiliza voto simples para medir a distância entre os objetos e designá-los ao agrupamento específico. Uma vantagem do agrupamento demográfico em relação ao k-médias consiste em que o primeiro possui habilidade para determinar automaticamente o número de grupos a serem gerados.

A tabela 3.2 apresenta as técnicas estudadas e suas características de acordo com os critérios utilizados.

TABELA 3.2 – Comparação de técnicas de MD.

Critério / Técnica	Modelo	Tarefa	Tipo de Aprendizado	Formato de Representação do Conhecimento
Árvore de Decisão	Preditivo	Classificação e Análise de associação	Supervisionado/ em lote	Árvore de decisão e Regras de classificação
Identificação de Regras Associativas	Descritivo e preditivo (com ressalvas)	Análise de associação	Não-supervisionado/ incremental	Regras associativas
Backpropagation	Preditivo	Classificação	Supervisionado/ em lote	Caixa preta
Modelo Neural Combinatório	Preditivo	Classificação e Análise de associação	Supervisionado/ incremental	Regras de classificação
Näive-Bayes	Preditivo	Classificação e Análise de associação	Supervisionado/ em lote	Regras de classificação
K-médias	Descritivo	Agrupamento	Não-supervisionado/ incremental	Centróide
Agrupamento Demográfico	Descritivo	Agrupamento	Não-Supervisionado/ em lote	Centróide

3.3 Pós-processamento

Nesta fase do processo de DCBD, os padrões identificados durante a MD são avaliados quanto à sua utilidade.

Um sistema de MD é capaz de gerar centenas ou até milhares de padrões ou regras, dos quais apenas uma pequena fração é realmente de interesse. Por esse motivo, na etapa de pós-processamento os padrões ou regras identificados são interpretados e compreendidos, a fim de verificar quais são realmente úteis.

Segundo Han [HAN2001], são considerados de interesse padrões que:
- validam uma hipótese que o usuário deseja confirmar;
- são facilmente compreendidos por humanos;
- são válidos para novos dados com algum grau de certeza;
- são potencialmente úteis; e
- são novos.

Nessa etapa do processo também é verificado se os padrões detectados são genéricos o suficiente ou se são válidos apenas para a base de dados utilizada na mineração. Além disso, é avaliado se os padrões são válidos para os objetivos da aplicação e se apresentam informação útil e nova.

A fim de verificar o quão interessante é cada regra ou padrão identificado, são utilizadas medidas objetivas, as quais são geralmente baseadas na estrutura dos padrões e em estatísticas a ela inerentes. Em geral, essas medidas são verificadas pelos próprios algoritmos de MD e cada uma está associada a um valor mínimo definido pelo usuário. Todavia, medidas objetivas, isoladamente, são insuficientes para identificar padrões interessantes, sendo necessário que sejam combinadas com medidas subjetivas, as quais refletem as necessidades e interesses particulares do usuário. Medidas subjetivas são baseadas no conhecimento do especialista a respeito do domínio da aplicação e no quê o

usuário espera obter a partir dos dados. Alguns padrões considerados interessantes com base em medidas objetivas podem ser descartados pelo especialista [HAN2001].

Além da validação dos padrões identificados, nessa etapa do processo de DCBD as regras e modelos extraídos podem ser traduzidos para uma linguagem inteligível pelo usuário, caso não sejam diretamente aplicáveis aos objetivos do mesmo.

Após a apresentação de uma visão geral do processo de DCBD de modo genérico, os próximos dois capítulos tratam a aplicação do processo na identificação de padrões de análise. Para tanto, são verificadas características específicas da aplicação, as quais devem ser levadas em consideração durante cada uma das fases do processo.

4 Aplicando DCBD: Preparação e Mineração de Esquemas de BDG

Segundo Buschman [BUS96] e Fowler [FOW97], a identificação de padrões para reuso requer o conhecimento de vários projetos e envolve a localização de uma solução eficiente, utilizada para resolver um problema recorrente. Os detalhes específicos da aplicação são abstraídos da solução reutilizada nos vários projetos. Essa construção resultante do processo de abstração é aplicada a novos projetos e testada, para, posteriormente, ser considerada um padrão.

Durante esse processo, com o objetivo de diminuir a interferência do especialista, acredita-se que a DCBD possa ser utilizada para auxiliar na localização de soluções recorrentes através da análise (semi)automatizada de esquemas conceituais de BDG. Dessa forma, pode ser considerado um número muito maior de esquemas em relação ao que vem sendo realizado pelos métodos atualmente conhecidos, possibilitando a análise conjunta de esquemas criados por diferentes especialistas. Essa característica, por conseqüência, potencializa a identificação de padrões baseados na experiência de um número maior de especialistas se comparados aos padrões que vêm sendo atualmente propostos.

Utilizar apenas esquemas conceituais de BDG como base de conhecimento para identificar padrões de análise, todavia, não possibilita a verificação de todos os elementos necessários à descrição de padrões, ainda que sejam consideradas as formas de descrição mais simples. Conforme visto na seção 2.2, a descrição de um padrão requer, no mínimo, a definição de um problema ou contexto e, posteriormente, a explicação de como tal problema pode ser resolvido. Sendo assim, a aplicação do processo de DCBD visa apontar soluções reutilizadas que, posteriormente analisadas por especialistas, podem ser eleitas padrões. Assim como ocorre tradicionalmente, um especialista no domínio da aplicação deve validar as construções freqüentes encontradas a partir do processo de DCBD, a fim de que estas sejam ou não eleitas padrões de análise. A essas construções freqüentes, ainda não validadas, convencionou-se chamar de *candidatos a padrão de análise*.

A fim de que o processo de DCBD seja aplicado na identificação de candidatos a padrão de análise, as atividades previstas para serem desenvolvidas no seu decorrer devem levar em consideração as peculiaridades inerentes a essa aplicação. Não faz parte do escopo desse trabalho explorar, em detalhes, todas as atividades do processo, algumas das quais são apresentadas na figura 4.1. A ênfase do trabalho corresponde às atividades mostradas em negrito na figura.

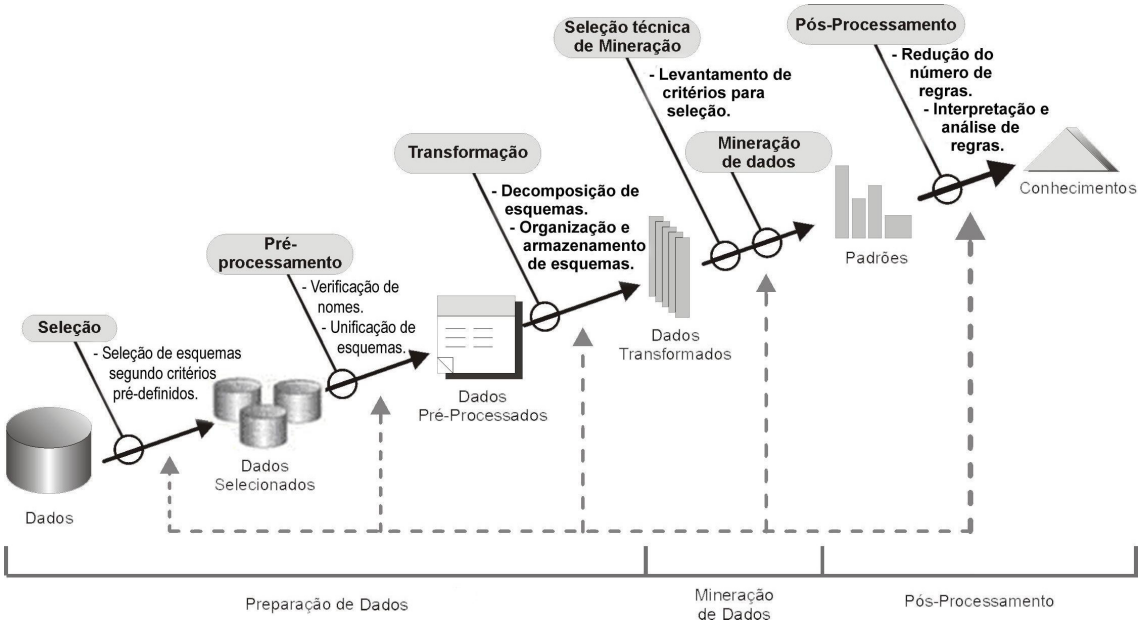


FIGURA 4.1 – Processo de DCBD para identificar candidatos a padrão de análise de BDG.

A escolha da técnica de MD a ser empregada durante o processo deve anteceder a preparação dos dados, pois seu conhecimento prévio é necessário à realização de algumas atividades dessa outra fase. Por exemplo, algumas técnicas de MD não manipulam valores contínuos, sendo necessário a sua discretização. No caso da mineração de esquemas de BDG, como eles não estão armazenados em um formato compatível com nenhuma das técnicas pesquisadas, o conhecimento prévio da técnica a ser aplicada permite o direcionamento desse armazenamento de modo a tirar maior proveito da aplicação da MD.

4.1 Mineração de Esquemas de BDG

O primeiro passo para a aplicação do processo de DCBD na identificação de candidatos a padrão de análise para BDG foi a escolha da técnica de MD a ser utilizada. Essa decisão foi tomada com base nos critérios apresentados na seção 3.2.1, os quais são analisados a seguir com base nas necessidades da aplicação.

a) Tipo de Modelo

A aplicação requer a geração de um modelo do tipo descritivo, pois visa à compreensão dos dados e não à previsão.

b) Tipo de aprendizado

O objetivo da utilização do processo de DCBD na identificação de candidatos a padrão de análise consiste em diminuir a interferência do ser humano no processo. Desse modo, devem ser aplicadas técnicas cuja dependência do especialista seja a menor possível. Tal requisito é atendido por técnicas que utilizem aprendizado do tipo não-supervisionado.

No que se refere ao volume de dados, não é possível mensurar a quantidade de esquemas de BDG que estarão disponíveis para mineração. Além disso, novos esquemas devem ser considerados e esquemas antigos podem ser atualizados. Nesse

contexto, uma técnica capaz de manipular os dados de entrada de forma incremental mostra-se mais adequada.

c) Tarefa a ser realizada

Acredita-se que a identificação de subesquemas recorrentes pode ser realizada a partir da identificação de associações freqüentes entre elementos de modelagem de esquema. Por exemplo, se é possível identificar que o relacionamento de continência entre um Pacote P e uma classe C ocorre frequentemente, dado um determinado conjunto de esquemas de BDG, então o subesquema formado pela classe juntamente com o pacote pode ser considerado recorrente no conjunto considerado.

Esse tipo de análise vem sendo realizado em outros domínios a partir da aplicação de técnicas de MD que realizam a tarefa de *análise de associações*, conforme descrito na seção 3.2.1.

d) Tipo de repositório a ser minerado

Apesar de consistir em um aspecto importante a ser avaliado durante a seleção da técnica de MD, essa característica não foi determinante no desenvolvimento deste trabalho, pois os esquemas de BDG não estão disponíveis em nenhum dos formatos atualmente aceitos por ferramentas de MD. Por outro lado, com base na técnica a ser aplicada, foi possível elaborar as estratégias de organização e armazenamento dos esquemas de BDG de modo a facilitar a MD, conforme é detalhado na seção 4.2.

e) Formato de representação do conhecimento

Para a identificação de candidatos a padrão de análise, o resultado da MD deve ser facilmente compreendido e é imprescindível que seja capaz de descrever padrões. Assim, é desejável que o formato de representação dos padrões seja do tipo caixa transparente.

O estudo das técnicas com base nos critérios acima descritos permite crer que a *identificação de regras associativas*, dentre as técnicas estudadas, é a que melhor se aplica na solução do problema de identificar padrões de análise para BDG a partir de esquemas conceituais. As principais características da técnica que levam a essa conclusão são:

- a técnica é baseada em aprendizado não-supervisionado, dispensando, portanto, a classificação prévia dos dados de entrada e a interferência do especialista nessa atividade;
- as regras associativas são facilmente compreendidas pelo ser humano (com sintaxe e semântica simplificadas), além de serem utilizadas para representar associações empíricas. Tais características as tornam uma linguagem apropriada para a descrição de padrões;
- existem algoritmos que manipulam os dados de entrada de forma incremental, ou seja, a inclusão de novos esquemas não implica na perda dos resultados já obtidos por minerações anteriores;
- pode-se garantir a completude da mineração através do uso de medidas de interesse [HAN2001]; e

- a tarefa realizada pela técnica, dentre as estudadas, é a mais adequada à solução do problema proposto.

A seção seguinte apresenta a técnica de identificação de regras associativas em detalhe.

4.1.1 Identificação de Regras Associativas e sua Aplicação na Identificação de Padrões para BDG

A técnica de identificação de regras associativas utiliza como dados de entrada um conjunto de transações, cada uma composta por um identificador e um conjunto de itens pertencentes ao domínio da aplicação. Dependendo da aplicação, os itens podem representar, por exemplo, produtos comprados em uma loja ou doenças apresentadas por um paciente. A partir desses dados, o algoritmo identifica a relação entre os itens. Por exemplo, no caso de produtos, poder-se-ia estar buscando aqueles que são comprados juntos. Já no caso das doenças, a busca poderia ser por aquelas de ocorrência simultânea.

No que se refere à organização dos dados, as transações podem ser dispostas de forma horizontal ou vertical [BRA98b].

Na organização horizontal (tabela 4.1), cada transação corresponde a uma linha do arquivo e existe uma coluna para cada item distinto. A organização vertical (tabela 4.2), por sua vez, dispõe cada item de transação em uma linha própria. O arquivo é composto por duas colunas, uma usada para identificar a transação e outra para discriminar o item. Nesse caso, a estrutura do arquivo (número de colunas) não é modificada quando um novo item é considerado. Por outro lado, uma mesma transação ocupa mais de uma linha e só pode ser recuperada através de seu identificador.

TABELA 4.1 – Exemplo de tabela com organizada horizontalmente.

Transação	Pão	Manteiga	Suco	Leite
1	P*	P	P	
2	P			P
3	P			
4	P	P		
5	P		P	

* P indica que o item está presente na transação

As ferramentas que implementam a técnica de identificação de regras associativas, em geral, trabalham com as duas formas de organização [BRA98b].

As relações identificadas entre os itens de transação são expressas em forma de regras associativas, formadas por itens do arquivo de entrada. O conseqüente das regras é um conjunto unitário [AGR93] ou um conjunto com $n > 0$ elementos [AGR94]. À cada regra identificada estão associados o suporte e a confiança, conforme descrito na seção 3.2.1.5.

As regras associativas são classificadas em vários tipos, segundo diferentes critérios**.**

TABELA 4.2 – Exemplo de tabela com organizada verticalmente.

Transação	Itens
1	Pão
1	Manteiga
1	Suco
2	Leite
2	Pão
3	Pão
4	Pão
4	Manteiga
5	Suco
5	Pão

Quanto aos tipos de valores presentes nas regras, elas são ditas quantitativas ou booleanas. As primeiras descrevem associações entre itens que apresentam valores quantitativos, os quais são particionados em intervalos. Regras booleanas, por sua vez, indicam a presença ou ausência de itens. A regra R_1 apresentada abaixo é um exemplo de regra quantitativa enquanto R_2 é uma regra booleana.

R_1: $30 \leq idade \leq 39$ *e* $4200 \leq renda \leq 4800 \rightarrow$ *compra = Tv de alta resolução*
R_2: *computador* $\rightarrow$ *impressora (*ou *compra = computador* $\rightarrow$ *compra = impressora)*

No que se refere à dimensão dos dados, as regras são classificadas em uni ou multidimensionais. Regras unidimensionais referem-se a uma única dimensão, como em R_2, onde a dimensão é o atributo *compra.* Regras multidimensionais, como R_1, referem-se a duas ou mais dimensões, nesse exemplo, os atributos *idade, renda* e *compra.*

Em [HAN95] e [SRI95] foi proposta a identificação de regras associativas baseadas em taxonomias, sendo possível encontrar regras em diferentes níveis de abstração. Por exemplo, "*computador* $\rightarrow$ *impressora"* e "*laptop* $\rightarrow$*zipdrive"*, *laptop* é uma especialização de computador e encontra-se em um nível de abstração mais baixo. Tais regras são importantes, pois, em alguns casos, as regras definidas em mais baixo nível de abstração não atingem os valores de suporte e/ou confiança mínimos, mas a regra mais genérica sim.

Outro tipo de regra bastante interessante são as regras associativas negativas [SAV98]**.** Regras desse tipo, ao contrário das convencionais, especificam quais itens não ocorrem em conjunto com outros. Assim, elas disponibilizam, por exemplo, informações do tipo "consumidores de cerveja não compram refrigerante".

Nesse trabalho foi utilizado o tipo mais simples de regra associativa que é a booleana unidimensional [HAN2001], pois seu poder de expressão é suficiente para a descrição de padrões de análise. Para identificar esse tipo de regra, Agrawal [AGR93] decompôs o problema em duas grandes fases:

- geração de <u>todas as combinações de itens</u> que possuem suporte acima do limite especificado. Tais combinações são chamadas de *large itemsets*.
- para um dado *large itemset* $Y = \{i_1, i_2,...,i_k | k>=2\}$, geração de todas as regras que usem itens do conjunto Y e cálculo do valor de sua confiança.

A etapa com maior custo computacional refere-se à geração dos conjuntos de itens freqüentes. Uma característica bastante importante para a qual Agrawal [AGR93] chama a atenção, nessa atividade, é que se Y é um *large itemset*, então todos os seus

subconjuntos não-vazios também o são (propriedade *Apriori*). Além disso, se o suporte de Y é superior ao limite especificado pelo usuário, todas as regras derivadas de Y também possuem suporte superior ao mínimo, pois são formadas, no máximo, por todos os itens em Y. Portanto, a obtenção da regra "R_3: A $\rightarrow$ B" não implica, necessariamente, em que a regra "R_4: A + C $\rightarrow$ B" também será obtida, pois o suporte desta última pode não atingir o mínimo especificado. Ainda para esse exemplo, se R_4 foi obtida com suporte s_1, então "A $\rightarrow$ B", com certeza, será gerada com suporte s_2, tal que $s_2 \geq s_1$, pois o conjunto de itens a partir do qual a regra R_3 foi obtida é um subconjunto daquele que resultou na regra R_4.

O suporte e a confiança são utilizados para fazer o *ranking* das regras geradas, sendo consideradas mais "fortes" aquelas com maior suporte e confiança.

Os princípios expostos acima são a base do algoritmo *apriori,* bastante utilizado na obtenção de regras associativas booleanas. Posteriormente à sua publicação, vários outros algoritmos [WEB2000, SRI96, ZAK2000, WAN2000] foram propostos com o objetivo de melhorar a performance do *apriori* e/ou gerar regras menos redundantes entre si.

O próprio algoritmo *apriori,* todavia, pode ser aplicado na identificação de padrões de análise para BDG. Nesse caso, o suporte da regra corresponde ao suporte do padrão por ela identificado, ou seja, ao percentual de ocorrência do padrão nos esquemas analisados. Sendo assim, o suporte mínimo especificado pelo usuário deve corresponder ao percentual mínimo em que uma construção deve estar presente nos esquemas analisados para que seja considerada um padrão. As construções pouco freqüentes (com valor de suporte inferior ao mínimo especificado) são eliminadas na fase de MD.

Embora bastante utilizada nas aplicações convencionais da técnica de identificação de regras associativas, a confiança não se mostra tão importante no contexto da identificação de candidatos a padrão de análise. Esse fato ocorre, pois, a confiança é uma medida utilizada para verificar o número de instâncias preditas corretamente pela regra e a tarefa de identificar padrões de análise não envolve predição. Além disso, a definição de padrão indica que uma construção pode ser classificada como tal se ela é bastante utilizada. Nesse contexto, não é relevante o fato de elementos dessa construção terem sido aplicados em alguns projetos de modo diferente do indicado pelo padrão. Por exemplo, se o esquema da figura 4.2 é freqüentemente reutilizado (e é considerado um padrão) é irrelevante o fato de que a classe "Lago" e o pacote "Hidrografia" tenham sido definidos separadamente em alguns dos esquemas analisados.

FIGURA 4.2 – Exemplo de padrão de análise

Uma vez que a confiança não é um parâmetro utilizado durante a identificação de padrões de análise, o valor mínimo especificado no algoritmo deve corresponder ao valor mínimo especificado para o suporte. Dessa forma, nenhuma regra é eliminada por não obter a confiança mínima.

4.2 Preparação de Esquemas de BDG

No contexto deste trabalho, algumas atividades da etapa de preparação de dados são tidas como já realizadas. Não obstante, uma visão geral das atividades mais importantes dessa fase do processo é apresentada a seguir. Nas próximas seções, são tratados com maior cuidado alguns aspectos específicos da aplicação, os quais foram estudados por serem considerados parte fundamental na verificação da adequação da técnica escolhida à aplicação em questão.

A seleção dos esquemas a serem minerados é a primeira atividade desta etapa e deve ser baseada em critérios específicos, a fim de que padrões não sejam desprezados devido a bases de conhecimento mal projetadas. Misturar esquemas de áreas de aplicações distintas pode acarretar a não identificação de padrões de áreas cuja amostragem é pequena.

Por outro lado, escalonar a mineração de acordo com uma área de aplicação específica pode impossibilitar a verificação de padrões formados por construções pertencentes a áreas de aplicação distintas. Por exemplo, considere uma base composta de 100 esquemas, onde 70 deles abordam aspectos referentes à hidrologia e 30 referem-se a sistema viário. Se o suporte mínimo para que uma estrutura seja aceita como candidata a padrão é de 50%, então nenhum subesquema utilizado na modelagem de sistema viário será classificado como tal, ainda que seja recorrente na maioria dos esquemas que descrevem esse aspecto da realidade. Por outro lado, se os esquemas das duas áreas de aplicação forem minerados isoladamente, não será possível identificar subesquemas formados por entidades de áreas distintas relacionadas entre si, como ruas e rios, por exemplo.

No que se refere ao pré-processamento e à limpeza de dados, um importante aspecto a ser observado consiste na verificação de nomes. Casos de sinônimos, antônimos e parônimos devem ser resolvidos, proporcionando a unificação de termos, para que candidatos a padrão sejam corretamente identificados.

No caso particular de esquemas conceituais de BDG, também se deve observar que os esquemas considerados podem ter sido criados com base em diferentes modelos de dados. Nesse caso, não é possível minerá-los em conjunto. Esse problema vem sendo tratado em [BAS2002], onde é proposta a criação de um conjunto união de conceitos, formado pelos construtores dos principais modelos de dados geográficos e cujo objetivo é unificar os esquemas quanto ao modelo de dados.

Também na fase de preparação de dados, os seguintes problemas devem ser tratados:

- a decomposição de esquemas em subesquemas menores, com maior probabilidade de serem considerados candidatos a padrão, e
- a organização dos subesquemas em um formato compatível com aquele utilizado pela técnica de MD escolhida.

Abaixo serão abordadas, em detalhe, as estratégias de decomposição de esquemas e a organização dos dados, uma vez que são atividades essenciais para que seja possível verificar a adequação da técnica de mineração à aplicação.

4.2.1 Decomposição de Esquemas

Para identificar candidatos a padrão de análise para BDG, através da técnica de identificação de regras associativas, esquemas conceituais de BDG são definidos como transações no arquivo de entrada.

Cada esquema conceitual é, recursivamente, decomposto em subesquemas, pois quanto menor o número de elementos que formam um subesquema, maior a chance dessa estrutura se repetir em esquemas de aplicações distintas. Cada subesquema resultante da decomposição corresponde a um item de transação no arquivo de entrada.

O processo de decomposição deve levar em consideração as regras de definição e de uso dos construtores do modelo de dados geográficos. Para facilitar a compreensão, os construtores dos modelos foram classificados em fortes e fracos. Construtores fortes são aqueles que, isoladamente, constituem um subesquema válido (que obedece as regras definidas no modelo de dados), como pacote e classe na UML, por exemplo. Construtores fracos, por sua vez, não existem sozinhos no esquema, ou seja, individualmente não formam subesquemas válidos. Sua definição dependente do seu relacionamento com um construtor forte, como ocorre, por exemplo, com atributos e associações. Desse modo, para ser considerado válido, um subesquema que contém uma instância de construtor fraco deve apresentar também a instância do construtor forte do qual a instância do construtor fraco depende.

Ainda que consideradas as regras de modelagem impostas pelos modelos de dados, o processo de decomposição gera um grande número de tipos de subesquema, sendo que nem todos são utilizados durante a mineração de dados. A definição dos tipos a serem considerados depende, basicamente, de três fatores:

- os construtores que formam os tipos de padrão a serem identificados;
- a granularidade dos padrões buscados; e
- a expressividade do formato de representação do conhecimento.

O primeiro aspecto guia a seleção dos subesquemas no sentido de desconsiderar aqueles que apresentam construtor não utilizado pelos tipos de padrão buscados. No escopo desse trabalho, é utilizado o subconjunto dos construtores do UML-GeoFrame [LIS2000] formado por pacote, classe, atributo definido em classe e associação binária simples. Esse subconjunto foi selecionado, pois se entende que ele compreende os construtores genéricos da orientação a objetos, utilizados na maioria, se não na totalidade, dos modelos de dados para BDG [BAS2002]. Os subesquemas são, ainda, caracterizados pelo fato de: atributos serem considerados apenas com nome, independente do domínio; associações binárias serem representadas sem cardinalidade; e a representação espacial dos objetos geográficos ser tratada como um atributo da classe.

Os tipos de padrão que se deseja identificar são definidos em diferentes níveis de granularidade, pois visam a auxiliar projetos baseados tanto na abordagem *bottom-up,* quanto na *top-down*. Sendo assim, os padrões podem ser formados por um ou vários elementos. No primeiro caso, são constituídos de uma instância de classe ou de pacote. Padrões formados por dois ou mais elementos os apresentam relacionados entre si através de associação binária entre classes, relação de continência entre pacote e classe ou relação de dependência entre atributo e classe.

A granularidade dos padrões e a expressividade do formato de saída determinam os subesquemas a serem considerados de acordo com o grau de desagregação dos elementos que os formam. Um padrão é tanto mais simples quanto menor for o número de elementos que o compõem. A medida em que aumenta a variação de granularidade entre o tipo de padrão mais simples e o mais complexo, maior é a quantidade de tipos de subesquema a serem processados. Por outro lado, quanto maior a expressividade do formato de representação do conhecimento, menor esse número.

As regras associativas permitem expressar associações empíricas entre os itens de transação. Além disso, por possuírem um grande poder de expressão, para identificar

os tipos de padrão buscados é suficiente considerar apenas alguns subesquemas formados por um, dois ou três elementos. Subesquemas não considerados nessa fase, assim como aqueles mais complexos, podem ser inferidos com base nas relações entre essas estruturas mais simples. Tais relações são identificadas durante a MD e expressas através das regras geradas, de modo que o conjunto dos tipos de subesquema considerado após a decomposição é o mínimo suficiente e necessário o qual permite identificar todos os tipos de padrão buscados.

Analisados os aspectos importantes referentes à decomposição de esquemas, segue a descrição de cada tipo de subesquema utilizado. Tais tipos são denominados de acordo com os construtores que utilizam e com o número de instâncias de cada construtor. Os subesquemas estão separados em grupos definidos de acordo com o número de elementos que os formam.

a) Tipos de subesquemas com um elemento:

- Pacote$_{(1)}$: apresenta apenas uma instância do construtor pacote, representado independente da(s) classe(s) nele definida(s). Cada elemento pacote presente no esquema gera um subesquema desse tipo.
- Classe$_{(1)}$: apresenta uma instância de classe, representada independente do pacote onde está inserida ou dos atributos e associações que a caracterizam. É criado um subesquema desse tipo para cada elemento classe presente no esquema.

b) Tipos de subesquemas com dois elementos:

- Atributo$_{(1)}$-Classe: apresenta os elementos atributo existentes no esquema. Como esse construtor é do tipo fraco e, no caso deste trabalho, está sempre associado a uma classe, ele é especificado juntamente com a classe onde foi definido. Cada elemento atributo presente no esquema origina um subesquema desse tipo. São considerados apenas os subesquemas formados por classe e um único atributo. Subesquemas compostos por classe com mais de um atributo não são considerados para efeito de MD, mas podem ser inferidos a partir da interpretação das regras na etapa de pós-processamento.
- Classe$_{(1)}$-Pacote: apresenta uma instância de classe, juntamente com o pacote no qual ela está contida. Esse tipo de subesquema possibilita a identificação de padrões formados por classes contidas em pacote, conforme será detalhado no capítulo 5. Cada classe definida dentro de um pacote gera um subesquema desse tipo. Subesquemas constituídos por várias classes pertencentes a um mesmo pacote não são considerados nessa fase, sendo obtidos, *aposteriori*, através da interpretação das regras.

c) Tipos de subesquemas com três elementos:

- Associação: apresenta as associações binárias. Como esse é um construtor fraco, uma associação é sempre definida juntamente com as classes que a forma. Cada associação binária do esquema gera um subesquema desse tipo. Auto-relacionamentos do tipo associação binária fazem parte deste grupo.
- Atributo$_{(1)}$-Classe-Pacote: apresenta um elemento atributo, agregando a informação do pacote que contém a classe onde esse atributo foi definido. Existe um subesquema desse tipo para cada atributo definido em classe contida em pacote. Assim como no tipo Atributo$_{(1)}$-Classe, apenas são considerados os subesquemas formados por um único atributo. Esse tipo de

subesquema é utilizado para identificar tanto os atributos das classes quanto o pacote no qual cada uma delas está definida.

A tabela 4.3 apresenta os subesquemas gerados com base no esquema da figura 4.2.

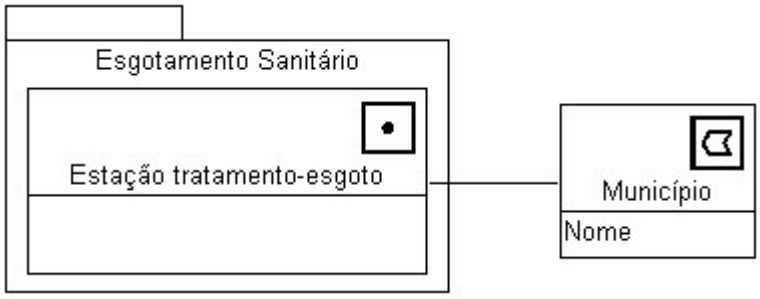


FIGURA 4.2 – Exemplo de esquema conceitual, baseado no UML-GeoFrame.

TABELA 4.3 – Subesquemas considerados após a decomposição do esquema da figura 4.2.

Tipo de Subesquema	Subesquemas Resultantes da Decomposição
$Pacote_{(1)}$	- Pacote "Esgotamento Sanitário"
$Classe_{(1)}$	- Classe "Estação tratamento-esgoto" - Classe "Município"
$Atributo_{(1)}$-classe	- Atributo "Polígono" na classe "Município" - Atributo "Nome" na classe "Município" - Atributo "Ponto" na classe "Estação tratamento-esgoto"
Associação	- Classe "Estação tratamento-esgoto" associada à classe "Município".
$Classe_{(1)}$-pacote	- Classe "Estação tratamento-esgoto" no pacote "Esgotamento Sanitário"
$Atributo_{(1)}$-classe-pacote	- Atributo "Ponto" na classe "Estação tratamento-esgoto", no pacote "Esgotamento Sanitário".

4.2.2 Armazenamento e Organização dos Esquemas

Usualmente, os algoritmos que implementam a técnica de identificação de regras associativas acessam os dados de entrada em arquivos planos ou tabelas. O formato tabular dos bancos de dados relacionais mostra-se ineficiente, pois as regras associativas geralmente são utilizadas em situações onde o domínio dos atributos é binário (o atributo está presente ou não) e apenas uma minoria dos atributos está presente em cada transação [WIT2000].

Em função dessas características e do fato de que é irrelevante, para o objetivo deste trabalho, armazenar os esquemas de maneira eficiente para serem consultados no futuro, decidiu-se por armazenar os esquemas de BDG em um arquivo plano (*flat file)* para fins de mineração. Cada esquema é armazenado nesse arquivo como uma seqüência de registros. O arquivo é formado por vários esquemas, cada um correspondendo a uma transação. Cada transação é composta por um ou mais itens de transação e por uma "marca de esquema". Os itens de transação consistem em subesquemas resultantes da decomposição do esquema original. A "marca de esquema"

é utilizada para facilitar a interpretação das regras, no sentido de identificar padrões formados por um único elemento, conforme detalhado na seção 5.1. O formato do arquivo, com base no formalismo EBNF, é mostrado a seguir.

ARQ ::= ESQ {,ESQ};
ESQ ::= TRANS;
TRANS ::= MARCAESQ, ITEMTRANS {,ITEMTRANS};
ITEMTRANS ::= SUBESQ;

No que se refere à organização dos dados para o caso específico da mineração de esquemas de BDG, sugere-se que a organização vertical seja adotada em virtude de sua flexibilidade e da grande quantidade de itens gerados após a decomposição de esquemas.

Outra preocupação quando se trata de organizar e armazenar esquemas de BDG para serem minerados, consiste em garantir a semântica de cada elemento, de modo a preservar suas características no decorrer do processo. Deve ser possível, ao obter o resultado da mineração, identificar o tipo de cada elemento. Para isso, os dados de entrada são descritos segundo notação própria, a qual é mostrada a seguir, com base na EBNF:

MARCAESQ ::= "E*";
SUBESQ ::= $Pacote_{(1)}$ | $Classe_{(1)}$| $Atributo_{(1)}$-Classe | Associação | $Classe_{(1)}$-Pacote | $Atributo_{(1)}$-Classe-Pacote;
$Pacote_{(1)}$ ::= <nome_pacote>":";
$Classe_{(1)}$ ::= <nome_classe>;
$Atributo_{(1)}$-Classe ::= ": "<nome_classe>": "<nome_atributo>;
Associação ::= <nome_classe>"#"<nome_classe>"#"<nome_associação>;
$Classe_{(1)}$-Pacote ::= <nome_pacote>": "<nome_classe>;
$Atributo_{(1)}$-Classe-Pacote ::= <nome_pacote>": "<nome_classe>":" <nome_atributo>;

Um exemplo de esquema, no formato do arquivo de entrada para mineração, é mostrado na tabela 4.4, com base no subesquema da figura 4.2.

TABELA 4.4 – Armazenamento do esquema da figura 4.2.

Id	Item
2	Esgotamento Sanitário:
2	Estação tratamento-esgoto
2	Município
2	: Município: Polígono
2	: Município: Nome
2	: Estação tratamento-esgoto: Ponto
2	Estação tratamento-esgoto#Município#Assoc [1]
2	Esgotamento Sanitário: Estação tratamento-esgoto
2	Esgotamento Sanitário: Estação tratamento-esgoto: Ponto
2	E*

[1] Uma vez que o nome da associação não foi definido no esquema, é representada apenas a sua ocorrência.

Após terem sido armazenados no formato proposto, os esquemas de BDG são submetidos a uma ferramenta de MD que gera regras associativas, tal como o IBM Intelligent Miner for Data. O resultado da mineração, por sua vez, consiste em um conjunto de regras associativas, tais como R_5, por exemplo, formadas pelos subesquemas apresentados no arquivo de entrada. Essas regras são, então, pós-processadas de acordo com os critérios apresentados no próximo capítulo.

R_5: Município $\rightarrow$: Município: Polígono

5 Aplicando DCBD: Pós-Processamento de Regras Associativas

Na etapa de pós-processamento, as regras identificadas durante a mineração de dados são interpretadas e avaliadas quanto à sua relevância para o domínio da aplicação. Para tanto, é de fundamental importância o auxílio de um especialista que examine, manualmente, as regras associativas geradas, verificando quais são realmente significativas [WIT2000].

Um problema inerente ao processo de mineração consiste em que os algoritmos que implementam a técnica de identificação de regras associativas tendem a encontrar um número excessivo de regras, tornando a avaliação do resultado quase tão trabalhosa quanto a própria mineração dos dados [AGR94]. No que diz respeito à aplicação específica de identificar candidatos a padrão de análise para BDG, verificou-se que este problema é ainda maior. Isso ocorre, porque cada padrão formado por mais de um elemento apresenta, implicitamente, outros padrões de granularidades menores, identificáveis a partir da decomposição do padrão inicial. O padrão da figura 5.1, por exemplo, apresenta implicitamente os padrões mostrados na figura 5.2, entre outros. Devido a essa característica, além de serem geradas regras que levam à identificação do padrão mais complexo, também são geradas aquelas que permitem inferir todos os seus subesquemas válidos, os quais são tão ou mais freqüentes do que o padrão mais complexo.

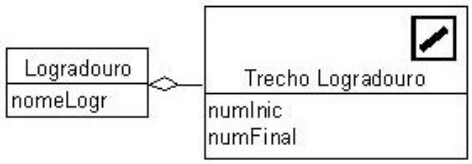


FIGURA 5.1 – Exemplo de padrão formado por mais de um elemento

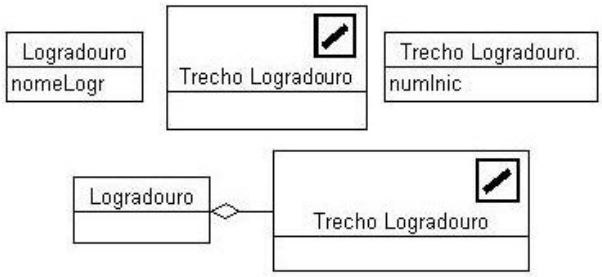


FIGURA 5.2 – Alguns padrões implícitos no padrão apresentado na figura 5.1.

Outra característica própria da aplicação e que contribui para a geração de regras desnecessárias consiste na repetição de elementos nos vários tipos de subesquemas considerados. Tal problema ocorre, por exemplo, entre subesquemas dos tipos $Classe_{(1)}$ e $Atributo_{(1)}$-Classe. Sempre que um subesquema do tipo $Atributo_{(1)}$-Classe estiver presente em uma regra, existirá pelo menos uma outra regra formada por ele juntamente com o subesquema do tipo $Classe_{(1)}$ que representa a instância da classe em questão. Desse modo, quanto maior o número de tipos de subesquemas contidos em outros, maior o número de regras identificadas durante a MD. Optar pelos subesquemas mais complexos ou pelos mais simples como alternativa para resolver o problema não é viável, pois se algum desses tipos de subesquemas não é considerado, não é possível a

identificação de alguns tipos de padrão buscados. Dentre os tipos de subesquema considerados, aqueles que contêm outros são indicados a seguir.

a) Classe$_{(1)}$-Pacote:

Esse tipo de subesquema contém os tipos Classe$_{(1)}$ e Pacote$_{(1)}$, os quais são essenciais na identificação de padrões formados por um único elemento, não podendo ser descartados.

Se não forem utilizados subesquemas do tipo Classe$_{(1)}$-Pacote, não é possível afirmar se a(s) classe(s) que compõe(m) o padrão está(ão) descrita(s) em um pacote. A partir de regras que combinam subesquemas dos tipos Classe$_{(1)}$ e Pacote$_{(1)}$ é possível verificar quais instâncias desses construtores são freqüentemente utilizadas em conjunto, não sendo possível garantir, porém, o relacionamento de continência entre elas.

b) Atributo$_{(1)}$-Classe:

Subesquemas desse tipo contêm aqueles do tipo Classe$_{(1)}$, os quais, conforme explicação no item "a", não podem ser desprezados.

Subesquemas do tipo Atributo$_{(1)}$-Classe, por sua vez, são o tipo mais simples que permite identificar atributos em classes.

c) Atributo$_{(1)}$-Classe-Pacote:

Contém subesquemas dos tipos Atributo$_{(1)}$-Classe e Classe$_{(1)}$-Pacote, além de Classe$_{(1)}$ e Pacote$_{(1)}$. A utilização de subesquemas desse tipo é importante, pois ele apresenta tanto o relacionamento de continência entre classe e pacote quanto o relacionamento de dependência entre atributo e classe. Através desse tipo de subesquema, é possível identificar além do(s) atributo(s) que caracteriza(m) as classes, o pacote ao qual cada classe pertence. Esse conhecimento não pode ser obtido a partir da combinação de subesquemas dos tipos Atributo$_{(1)}$-Classe e Classe$_{(1)}$-Pacote, pois, nesse caso, não é possível garantir que as ocorrências da classe no pacote e do atributo na classe são válidas em um mesmo esquema.

Uma vez que não foi possível reduzir nem o número de subesquemas considerados nem o número de regras geradas, os tipos de regra resultantes ao final da mineração foram analisados, trabalho parcialmente disponível em [SIL2001]. Com base nesse estudo foi possível verificar que:

- apesar de cada padrão ser identificável por uma única regra, cada padrão está contido em várias regras;
- um subconjunto das regras geradas é suficiente para identificar todos os candidatos a padrão existentes nos esquemas minerados; e
- algumas regras estão contidas em outras.

Com base nessas constatações e no conhecimento dos tipos de padrão que se deseja identificar, foi possível isolar dois subconjuntos de regras. O primeiro formado por regras úteis à identificação de padrões e o segundo, por aquelas que, comprovadamente, não levam à inferência de nenhum padrão buscado. A partir dessa atividade foi possível reduzir, automaticamente, na fase de pós-processamento, a quantidade de regras a serem posteriormente avaliadas pelo especialista humano.

Conforme citado na seção 4.1.1, nessa fase do processo são consideradas apenas as regras cujo suporte e confiança são maiores que os respectivos valores mínimos

especificados pelo usuário. Portanto, todas as regras são formadas por elementos freqüentes nos esquemas analisados. Todavia, conforme anteriormente citado, a análise das regras geradas indica que nem todas são úteis à identificação de candidatos a padrão. O subconjunto das regras úteis foi denominado "regras de interesse".

Assim, a fase de pós-processamento das regras associativas visando à identificação de candidatos a padrão de análise para BDG consiste em:

- destacar regras de interesse da aplicação;
- eliminar, de forma automática, regras, comprovadamente, não relevantes; e
- apresentar ao especialista humano o resultado dos passos anteriores, para que ele confirme ou não os candidatos a padrão identificados pelas regras de interesse e avalie as regras restantes quanto à sua utilidade.

Parte do processo de seleção das regras de interesse consiste em identificar quais tipos de regras, de fato, podem ser utilizados na identificação dos padrões buscados. Essa atividade se mostra importante, pois permite a eliminação de regras que, apesar de serem formadas por construções freqüentes, não levam a padrões úteis ou relevantes dentro do que é proposto neste trabalho.

Para reduzir a quantidade de regras, foram verificados, em primeiro lugar, os tipos de padrão que se deseja encontrar (aqueles constituídos de uma instância de classe ou de pacote; ou aqueles formados por dois ou mais elementos relacionados entre si). Para cada tipo de padrão, verificou-se uma forma intuitiva de descrevê-lo no formato de regras associativas. Em seguida, esse formato foi generalizado, tornando-se um tipo de regra de interesse. Identificadas as regras de interesse, foram verificados alguns tipos de regra que, comprovadamente, não são úteis à identificação de nenhum dos padrões buscados. Com base no resultado desse trabalho, foi possível, então, automatizar a eliminação das regras irrelevantes, de modo que o especialista pode facilmente identificar as regras de interesse no conjunto restante.

No contexto desse trabalho, para simplificar tanto a interpretação quanto a validação das regras, são consideradas apenas aquelas cujo conseqüente é formado por um único item, conforme descrito em [AGR93]. Nas próximas seções, são apresentados o conjunto de regras de interesse, assim como a estratégia de redução do número de regras.

5.1 Caracterização do Conjunto das Regras de Interesse

Durante a análise das regras para a identificação dos tipos de regra de interesse, verificou-se que a estratégia de decomposição e organização de esquemas proposta não favorece o processo de inferência de candidatos a padrão formados por um único elemento (instância de pacote ou classe), não tendo sido possível identificar nenhuma forma intuitiva de combinar os subesquemas considerados de modo a inferir os tipos de padrão em questão. Como alternativa para minimizar esse problema, o item "marca de esquema" foi incorporado ao arquivo de entrada.

A "marca de esquema" não representa um subesquema gerado a partir da decomposição dos esquemas de BDG, mas visa registrar a ocorrência de um esquema qualquer. Todo esquema apresentado na entrada possui um item correspondente a "marca de esquema". Assim, o suporte das regras cujo antecedente é formado apenas por esse item indica o percentual de esquemas analisados que contém a construção presente no seu conseqüente.

Se, por um lado, a "marca de esquema" facilita a identificação de padrões formados por um único elemento, sua incorporação obrigatória ao arquivo de entrada

implica em aumento significativo no número de regras geradas na MD, pois, para cada padrão identificado, várias regras contendo esse elemento são geradas desnecessariamente.

Assim, para ser considerada de interesse, a regra deve atender a dois requisitos que visam tornar mais natural tanto a leitura quanto a interpretação das mesmas, além de restringir o número de regras a ser analisado pelo especialista. Os requisitos são:

- R1: o padrão identificado pela regra é formado apenas por construtores presentes no seu conseqüente, exceto quando o conseqüente for um subesquema do tipo Associação. Nesse caso, a regra também pode ser utilizada para identificar padrões mais complexos; e
- R2: elementos fortes (instâncias de construtor forte) que formam o subesquema do conseqüente devem pertencer ao conjunto dos elementos presentes no antecedente da regra, exceto quando o conseqüente for formado por um subesquema do tipo $Classe_{(1)}$ ou $Pacote_{(1)}$. Nesse caso, o antecedente da regra deve conter apenas o item "marca de esquema".

O requisito R1 visa padronizar o tipo de regra a ser utilizado na identificação de cada tipo de padrão buscado. Com base nele, é possível manter apenas uma das regras entre aquelas formadas pelos mesmos elementos de esquema, como R_6 e R_7, por exemplo. Nesse caso, as duas regras poderiam ser utilizadas para identificar o padrão formado pela classe Município qualificada pelo atributo Polígono. A definição do requisito R1, todavia, permite a eliminação da regra R_7 e garante que, ainda assim, o padrão será identificado (pela regra R_6).

R_6: Município $\rightarrow$: Município: Polígono
R_7: : Município: Polígono $\rightarrow$ Município

A tabela 5.1 mostra um resumo dos tipos de padrão identificáveis com base no tipo de subesquema presente no conseqüente da regra.

TABELA 5.1 – Relação entre tipo de padrão e subesquema do conseqüente das regras.

Tipo de subesquema do conseqüente da regra	**Estrutura do tipo de candidato a padrão identificado**
$Classe_{(1)}$	Apenas uma instância de classe.
$Pacote_{(1)}$	Apenas uma instância de pacote.
$Atributo_{(1)}$-Classe	Uma instância de classe com atributo(s).
$Classe_{(1)}$-Pacote	Uma instância de pacote contendo uma ou mais classes sem atributos e sem associações entre si.
$Atributo_{(1)}$-Classe-Pacote	Uma instância de pacote com uma ou mais classes e seu(s) respectivo(s) atributo(s). As classes não possuem associação entre si.
Associação	Duas ou mais instâncias de classe relacionadas através de associação(ões) binária(s). As classes podem ou não estar definidas em pacote e apresentar atributo(s).

R2, por sua vez, visa tanto tornar mais intuitiva a leitura das regras quanto garantir que os padrões identificados são formados por elementos relacionados entre si. Essa restrição não elimina regras de interesse, pois, para todos os conseqüentes formados por mais de um elemento (ou seja, que contêm um elemento fraco), com certeza, existe uma regra que apresenta os elementos fortes desse subesquema no seu

antecedente. Isso ocorre, pois a estratégia de decomposição dos esquemas garante que os elementos fortes, individualmente, também serão considerados na mineração e, por serem mais simples, estes são tão ou mais freqüentes que o subesquema que contém também o elemento fraco. Esse requisito implica em que, para inferir o padrão formado pela classe Município contendo o atributo Polígono, a regra R_6 (anteriormente apresentada) será utilizada no lugar da regra R_8, por exemplo.

R_8: Esgotamento Sanitário: $\rightarrow$: Município: Polígono

Ainda para efeito de interpretação, as regras foram classificadas em simples e complexas. Uma regra simples apresenta o candidato a padrão integralmente no seu conseqüente e não requer qualquer conhecimento especialista para a identificação dessa estrutura. Regras simples caracterizam-se por apresentar, no conseqüente, todos os elementos fortes presentes no seu antecedente. Além disso, não existe elemento fraco no antecedente de uma regra simples.

As regras complexas, por sua vez, podem apresentar elementos fracos no antecedente e nem todos os elementos fortes do antecedente estão presentes no conseqüente da regra. Sua interpretação combina informações tanto do antecedente quanto do conseqüente para identificar o candidato a padrão, exigindo a aplicação do conhecimento objetivo do especialista em projeto de BDG, ou seja, domínio dos construtores dos modelos de dados e das regras de combinação dos mesmos. A partir deste tipo de regra é possível inferir candidatos a padrão mais complexos do que o subesquema apresentado no conseqüente da regra e do que os subesquemas considerados durante a preparação dos dados.

A tabela 5.2 mostra a relação entre os tipos de regras e o conhecimento especialista necessário, tanto para sua interpretação quanto para validação dos candidatos a padrão.

TABELA 5.2 – Relação entre o tipo de regra e o tipo de conhecimento especialista.

Atividade **Tipo de regra**	**Interpretação de regras**	**Validação de candidatos a padrão**
Simples	Desnecessário o conhecimento do especialista	Conhecimento subjetivo do especialista
Complexa	Conhecimento objetivo do especialista	Conhecimento subjetivo do especialista

Na seção a seguir, são apresentados alguns tipos de regra de interesse, descritos a partir dos tipos de padrão que podem ser inferidos por eles. A estrutura das regras é apresentada segundo a EBNF, onde "S" é o suporte e "CF", a confiança da regra. Devido ao exposto no final da seção 4.1.1, a confiança não é considerada para fins de inferência do padrão. Nas regras apresentadas a seguir, o operador lógico "e" (*and*) é representado pelo símbolo "+".A informação que precede o ponto (.) corresponde ao tipo da regra e a informação que sucede esse símbolo indica a qual parte da regra (antecedente ou conseqüente) a descrição se refere. Os exemplos descrevem os itens na notação UML, onde a "marca de esquema" é representada conforme mostra a figura 5.3, sugerindo um esquema completo.

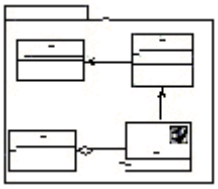


FIGURA 5.3 – Representação gráfica do elemento "Marca de esquema".

5.1.1 Interpretação de Regras Simples

A interpretação desse tipo de regra é bastante trivial e não requer conhecimento do especialista, uma vez que o padrão está completamente descrito no seu conseqüente. Abaixo são detalhados os tipos de regra simples identificados como de interesse. Para cada tipo é fornecido um exemplo de tipo de padrão que pode ser inferido a partir do tipo de regra em questão.

a) Tipo S1

Utilizado para identificar padrões formados por uma única instância de classe. Sua estrutura é:

S1.ANTEC::= MARCAESQ;

S1.CONSEQ ::= $Classe_{(1)}$; S% e CF%.

Esse tipo de regra indica que a instância da classe presente no seu conseqüente é utilizada em S% dos esquemas apresentados na entrada.

Um exemplo desse tipo de regra e o padrão por ela identificado podem ser visualizados, respectivamente, nas figuras 5.4 e 5.5.

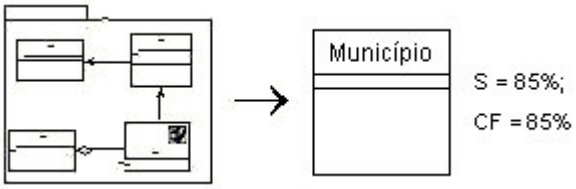


FIGURA 5.4 – Exemplo de regra simples tipo S1.

FIGURA 5.5 – Padrão inferido a partir da regra da figura 5.4.

b) Tipo S2

Permite inferir padrões formados por uma única instância de pacote, sua estrutura é mostrada a seguir.

S2.ANTEC::= MARCAESQ;

S2.CONSEQ ::= $Pacote_{(1)}$; S% e CF%.

Esse tipo de regra indica que a instância de pacote presente no seu conseqüente é utilizada em S% dos esquemas apresentados na entrada.

Exemplo desse tipo de regra, assim como o padrão por ela identificado, são mostrados, respectivamente, nas figuras 5.6 e 5.7.

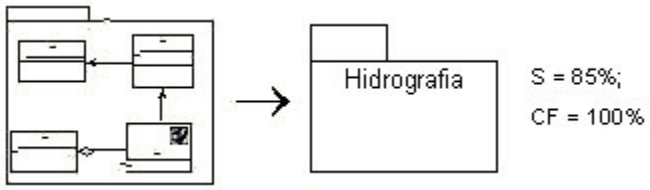


FIGURA 5.6 – Exemplo de regra simples tipo S2.

FIGURA 5.7 – Padrão inferido a partir da regra da figura 5.6.

c) Tipo S3

Identifica padrões formados por uma única instância de classe, qualificada por um único atributo. A estrutura da regra é:

S3.ANTEC::= Classe$_{(1)}$;

S3.CONSEQ ::= Atributo$_{(1)}$-classe; S% e CF%.

Esse tipo de regra indica que, em S% dos esquemas considerados, a classe do seu antecedente foi utilizada e qualificada pelo atributo apresentado no conseqüente.

Um exemplo desse tipo de regra e o padrão por ela identificado são apresentados, respectivamente, nas figuras 5.8 e 5.9.

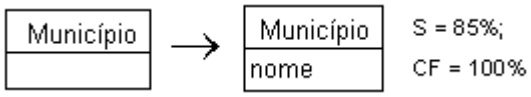


FIGURA 5.8 – Exemplo de regra simples tipo S3.

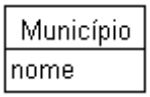


FIGURA 5.9 – Padrão inferido a partir da regra da figura 5.8.

d) Tipo S4

Utilizado para identificar padrões formados por uma única instância de classe, definida em um pacote. Sua estrutura é:

S4.ANTEC::= Pacote$_{(1)}$, "+", Classe $_{(1)}$;

S4.CONSEQ ::= Classe$_{(1)}$-Pacote; S% e CF%.

A partir desse tipo de regra é possível detectar que a classe e o pacote presentes no antecedente da regra são utilizados em S% dos esquemas. Além disso, a classe está contida no pacote.

Um exemplo desse tipo de regra e o padrão inferido são apresentados nas figuras 5.10 e 5.11.

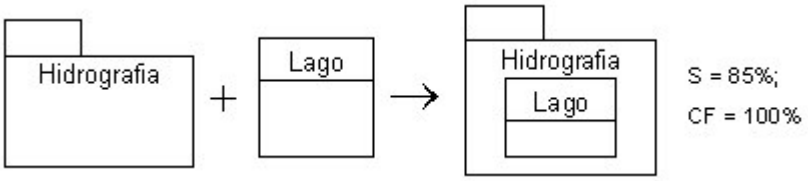


FIGURA 5.10 – Exemplo de regra simples tipo S4.

FIGURA 5.11 – Padrão inferido a partir da regra da figura 5.10.

e) Tipo S5

Permite a inferência de padrões formados por uma única instância de classe, qualificada por um único atributo, dentro de pacote. Sua estrutura é apresentada a seguir:

S5.ANTEC::= Classe$_{(1)}$-Pacote;

S5.CONSEQ ::= Atributo$_{(1)}$-Classe-Pacote; S% e CF%.

A regra indica que se a classe do antecedente está presente no esquema e é definida no pacote também apresentado no antecedente da regra, então ela é qualificada pelo atributo mostrado no conseqüente. Isso ocorre em S% dos esquemas apresentados na entrada.

Exemplo desse tipo de regra e o padrão identificado são mostrados nas figuras 5.12 e 5.13.

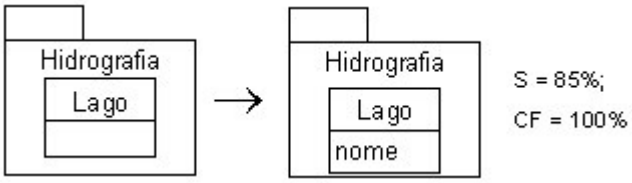


FIGURA 5.12 – Exemplo de regra simples tipo S5.

FIGURA 5.13 – Padrão inferido a partir da regra da figura 5.12.

f) Tipo S6

Utilizado para identificar padrões formados por duas classes relacionadas entre si através de uma associação binária. A estrutura desse tipo de regra é:

S6.ANTEC::= Classe$_{(1)}$, "+", Classe $_{(1)}$;

S6.CONSEQ ::= Associação; S% e CF%.

Esse tipo de regra indica que as classes do antecedente estão presentes em S% dos esquemas apresentados na entrada e, além disso, estão relacionadas através de uma associação binária.

Um exemplo desse tipo de regra e o padrão por ela identificado são mostrados nas figuras 5.14 e 5.15.

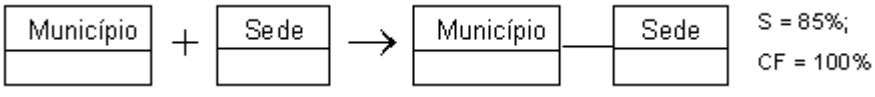


FIGURA 5.14 – Exemplo de regra simples tipo S6.

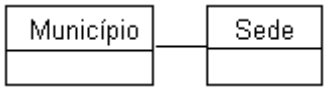


FIGURA 5.15 – Padrão inferido a partir da regra da figura 5.14.

g) Tipo S7

Utilizado para identificar padrões formados por uma classe com auto-relacionamento do tipo associação binária. A estrutura desse tipo de regra é:

S7.ANTEC::= $Classe_{(1)}$;

S7.CONSEQ ::= Associação; S% e CF%.

Esse tipo de regra indica que a classe do antecedente está presente em S% dos esquemas apresentados na entrada e, além disso, possui um auto-relacionamento do tipo associação.

Um exemplo desse tipo de regra e o padrão por ela identificado são mostrados nas figuras 5.16 e 5.17.

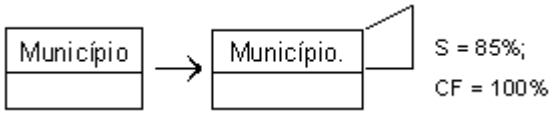


FIGURA 5.16 – Exemplo de regra simples tipo S7.

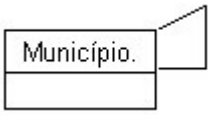


FIGURA 5.17 – Padrão inferido a partir da regra da figura 5.16.

5.1.2 Interpretação de Regras Complexas

Regras complexas são utilizadas para inferir padrões de interesse mais complexos do que os tipos de subesquema considerados durante a preparação de dados. Nesses tipos de regra, nem todos os elementos do antecedente estão presentes no conseqüente da regra. Para serem consideradas de interesse, as regra complexas devem atender a uma das condições:

- os elementos fortes mostrados no antecedente da regra estão presentes no seu conseqüente; ou
- os elementos fortes presentes no antecedente da regra estão relacionados entre si através de: associação binária entre classes; relação de continência entre pacote e classe; e/ou relação de dependência entre atributo e classe.

As condições acima descritas garantem que todos os elementos presentes na regra estão relacionados entre si de alguma maneira. Essa restrição é importante, pois despreza regras que apresentam elementos não relacionados a nenhum outro, as quais não são úteis à identificação dos padrões buscados. Tais regras contêm outras mais simples (com menos elementos) as quais permitem a inferência de tais padrões.

A seguir são apresentados alguns tipos de regras complexas identificados como de interesse. Outros tipos de regras complexas podem ser encontrados no Anexo 1.

a) Tipo C4c

Identifica padrões formados por duas ou mais classes não associadas, definidas em um mesmo pacote. Além da classe do conseqüente, outra(s) classe(s) apresenta(m) um ou mais atributos. Sua estrutura é:

C4c.ANTEC::= Classe$_{(1)}$-Pacote, {"+" Classe $_{(1)}$-Pacote}, "+" Atributo $_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote};

C4c.CONSEQ ::= Atributo$_{(1)}$-Classe-Pacote; S% e CF%.

Todos os pacotes devem ser iguais. Nesse caso, com exceção da classe do conseqüente, todas as classes que apresentam atributo devem ser apresentadas no antecedente apenas em subesquemas do tipo Atributo$_{(1)}$-Classe-Pacote. Se a classe do conseqüente possui mais de um atributo, deve ser apresentada no antecedente em item do tipo Atributo$_{(1)}$-Classe-Pacote e não em Classe$_{(1)}$-Pacote. Classes sem atributo ou classe do conseqüente com apenas um atributo são apresentadas no antecedente em item do tipo Classe$_{(1)}$-Pacote.

A partir desse tipo de regra é possível inferir que S% dos esquemas apresentados na entrada possuem o pacote especificado na regra, definido com as várias classes sem relacionamentos entre si. Além disso, as classes são qualificadas pelos atributos, conforme apresentados na regra.

Exemplo desse tipo de regra e o padrão por ela identificado são mostrados nas figuras 5.18 e 5.19.

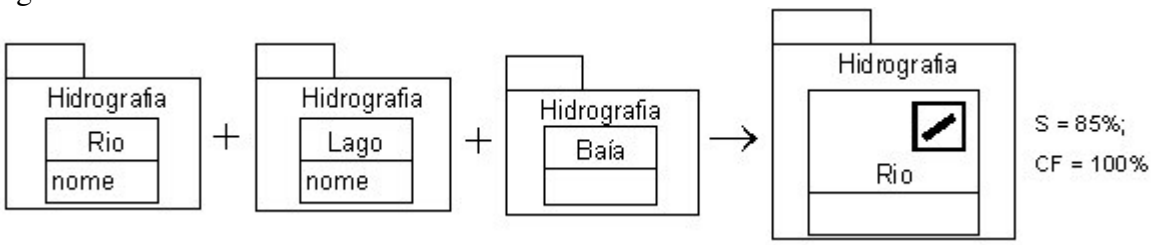


FIGURA 5.18 – Exemplo de regra complexa tipo C4c.

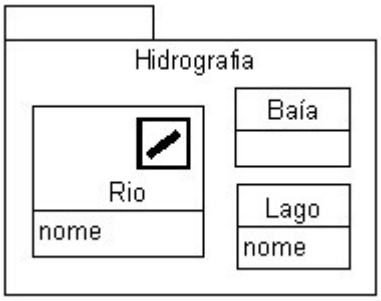


FIGURA 5.19 – Padrão inferido a partir da regra da figura 5.18.

b) Tipo C5a

Utilizado para identificar padrões formados por duas classes definidas em um mesmo pacote e relacionadas entre si através de associação binária.

C5a.ANTEC::= Classe$_{(1)}$-Pacote, "+", Classe $_{(1)}$-Pacote;

C5a.CONSEQ ::= Associação; S% e CF%.

Os pacotes, nesse tipo de regra, são obrigatoriamente iguais.

Esse tipo de regra indica que, em S% dos esquemas analisados, as classes do antecedente da regra estão relacionadas entre si através de uma associação binária. Além disso, as duas classes encontram-se definidas no pacote apresentado.

Um exemplo desse tipo de regra e o padrão por ela identificado são mostrados nas figuras 5.20 e 5.21.

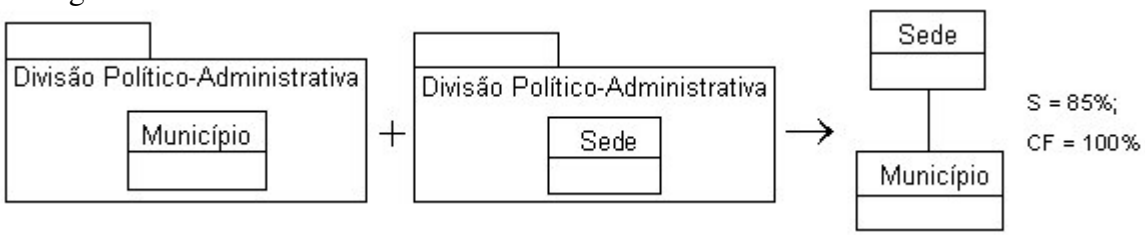


FIGURA 5.20 – Exemplo de regra complexa tipo C5a.

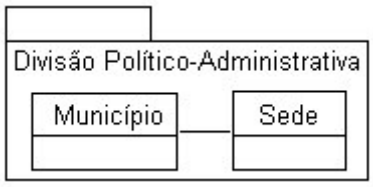


FIGURA 5.21 – Padrão inferido a partir da regra da figura 5.20.

c) Tipo C7

Utilizado para inferir padrões que consistem em duas classes relacionadas entre si através de uma associação, onde pelo menos uma das classes possui atributo e está definida em pacote.

C7.ANTEC::= Classe$_{(1)}$, "+", Atributo $_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote} | Classe$_{(1)}$-Pacote, "+", Atributo $_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote} | Atributo$_{(1)}$-Classe, "+", Atributo $_{(1)}$-Classe-Pacote, {"+" At ributo$_{(1)}$-Classe-Pacote} | Atributo$_{(1)}$-Classe-Pacote, "+" Atributo $_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote};

C7.CONSEQ ::= Associação; S% e CF%.

Esse tipo de regra indica que, se as classes do antecedente estão presentes no esquema, então existe um relacionamento de associação entre elas em S% dos esquemas. Além disso, as classes são definidas em pacotes e qualificadas por atributo(s), conforme descrito no antecedente da regra.

Exemplos desse tipo de regra e os padrões por elas identificados são mostrados nas figuras 5.22, 5.23, 5.24, 5.25, 5.26, 5.27, 5.28 e 5.29.

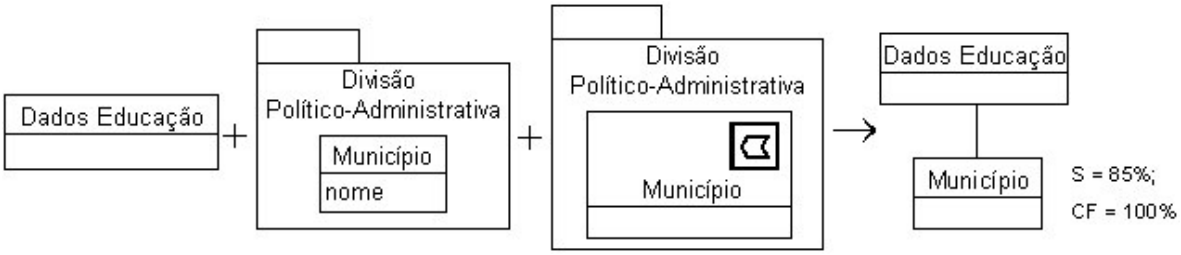


FIGURA 5.22 – Exemplo de regra complexa tipo C7.

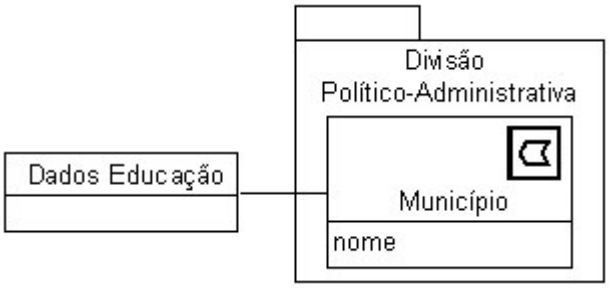


FIGURA 5.23 – Padrão inferido a partir da regra da figura 5.22.

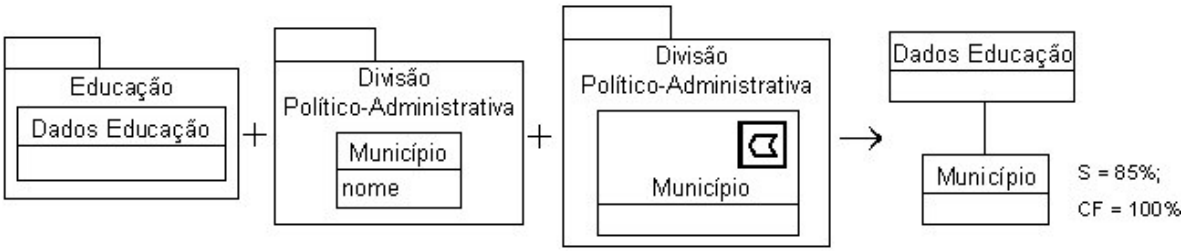


FIGURA 5.24 – Exemplo de regra complexa tipo C7.

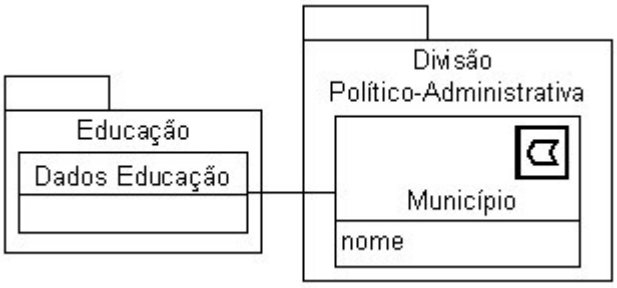


FIGURA 5.25 – Padrão inferido a partir da regra da figura 5.24.

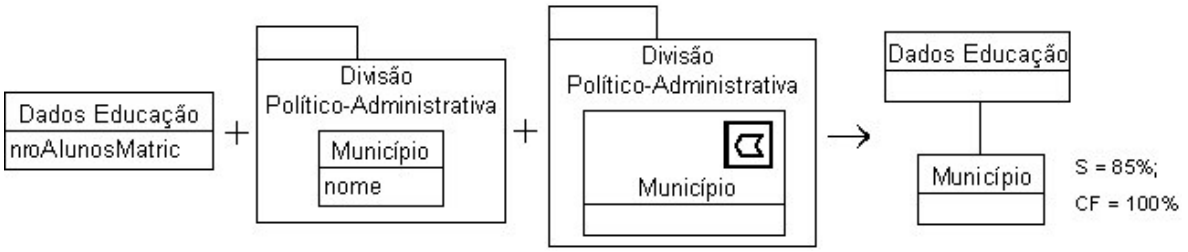


FIGURA 5.26 – Exemplo de regra complexa tipo C7.

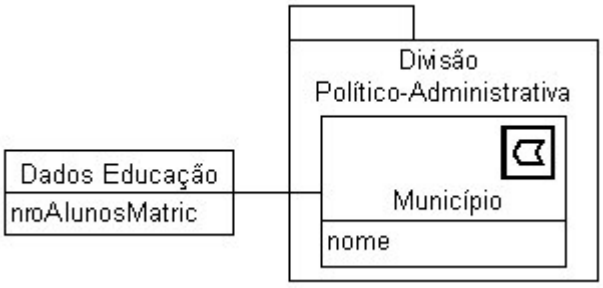


FIGURA 5.27 – Padrão inferido a partir da regra da figura 5.26.

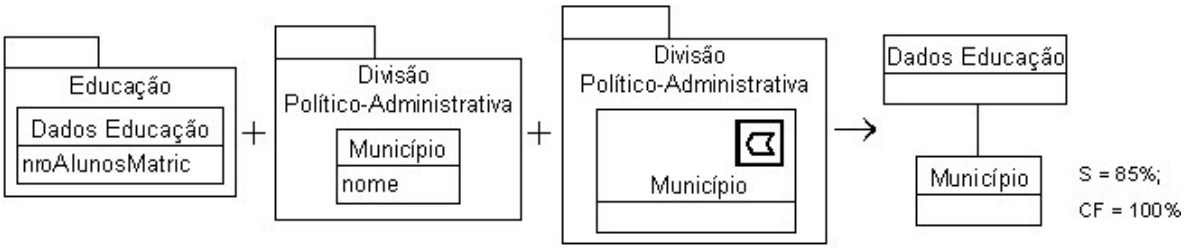


FIGURA 5.28 – Exemplo de regra complexa tipo C7.

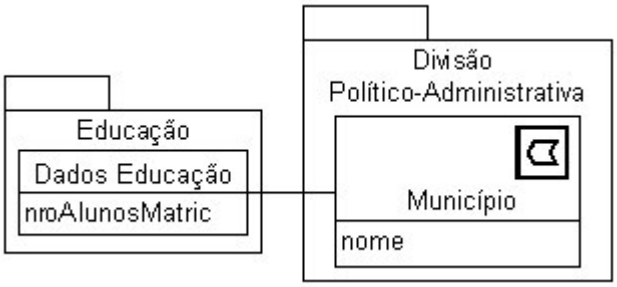


FIGURA 5.29 – Padrão inferido a partir da regra da figura 5.28.

d) Tipo C10c

Identifica padrões formados por várias classes associadas entre si, onde pelo menos uma delas é qualificada por, no mínimo, um atributo e uma delas é definida dentro de pacote. Inclui classe(s) com atributo(s), definida(s) em pacote.

C10c.ANTEC::= Classe$_{(1)}$-Pacote, "+", Atributo $_{(1)}$-Classe, "+", Associação, {"+" Associação}, {"+" Classe $_{(1)}$-Pacote}, {"+" Atributo $_{(1)}$-Classe}, {"+" Atributo $_{(1)}$-Classe-Pacote}| Classe$_{(1)}$, "+", Classe $_{(1)}$-Pacote, "+", Atributo $_{(1)}$-Classe, "+", Associação, {"+" Associação}, {"+" Classe $_{(1)}$-Pacote}, {"+" Atributo $_{(1)}$-Classe}, {"+" Atributo $_{(1)}$-Classe-Pacote};

C10c.CONSEQ ::= Associação; S% e CF%.

O item do tipo Classe$_{(1)}$ só deve estar presente na regra se uma das classes do conseqüente não possui outra associação, não é qualificada por atributo(s) e não está definida em pacote.

A leitura desse tipo de regra indica que em S% dos esquemas, as classes presentes na regra possuem associações entre si, são qualificadas por atributo(s) e definidas em pacote, conforme descrito na regra.

Exemplos desse tipo de regra e os padrões por elas identificados são mostrados nas figuras 5.30, 5.31, 5.32 e 5.33.

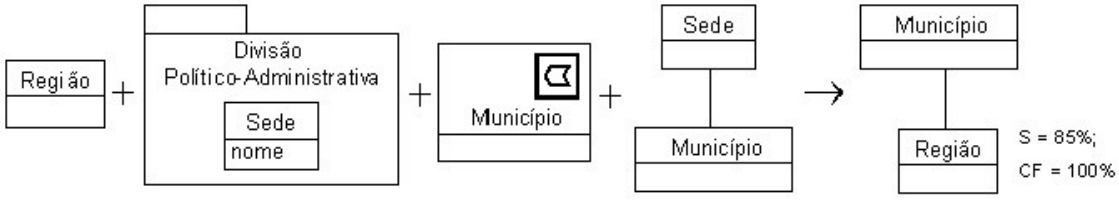


FIGURA 5.30 – Exemplo de regra complexa tipo C10c.

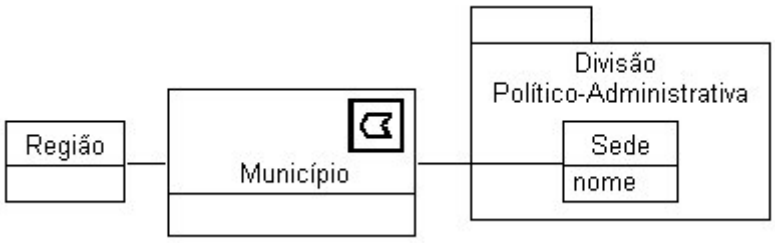


FIGURA 5.31 – Padrão inferido a partir da regra da figura 5.30.

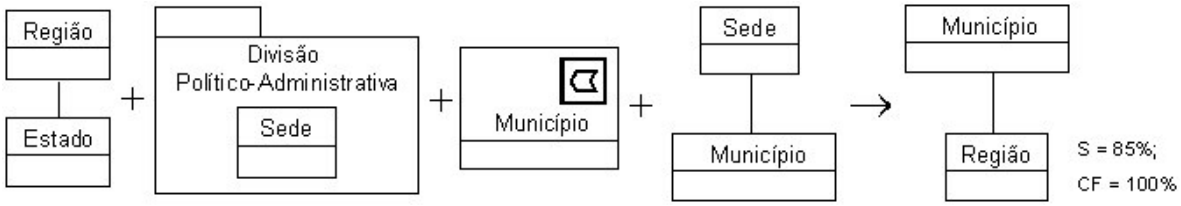


FIGURA 5.32 – Exemplo de regra complexa tipo C10c.

FIGURA 5.33 – Padrão inferido a partir da regra da figura 5.32.

5.2 Redução de Regras

A partir da identificação de regras de interesse, é possível isolar um conjunto de regras que não são úteis à identificação de nenhum dos tipos de padrão buscados. Para isso, são considerados tanto os requisitos definidos para facilitar a interpretação das regras quanto as características próprias da aplicação, as quais são responsáveis pela geração de regras redundantes ou sem significado. São consideradas desprezíveis regras que:

- não atendem ao requisito R2, apresentado na seção 5.1;
- não atendem a nenhuma das condições descritas na seção 5.1.2; ou
- apresentem no seu antecedente item(ns) que contém(êm) outro(s).

As regras que não atendem ao requisito R2 podem ser substituídas por outra(s) que atende(m), a(s) qual(is), com certeza, também fará(ao) parte do conjunto de regras geradas.

As que não atendem às condições da seção 5.1.2 são descartadas, pois os padrões buscados são formados sempre por um único elemento ou por mais de um elemento relacionados entre si.

As regras do terceiro grupo, por sua vez, contêm outras mais simples, as quais são suficientes para identificar o padrão buscado. Um exemplo de regra descartada por este motivo é apresentado na figura 5.34. A regra que a substitui é mostrada na figura 5.35 e o padrão buscado está ilustrado na figura 5.36.

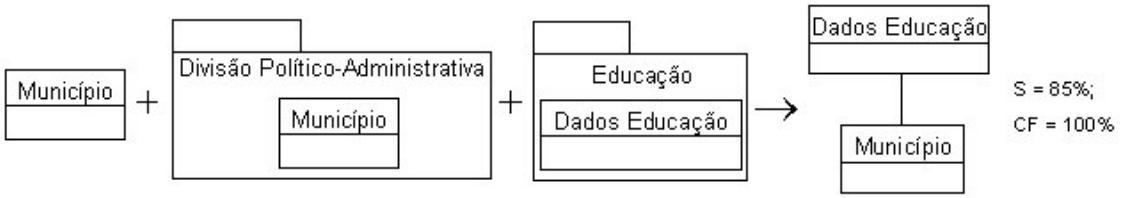


FIGURA 5.34 – Exemplo de regra descartada.

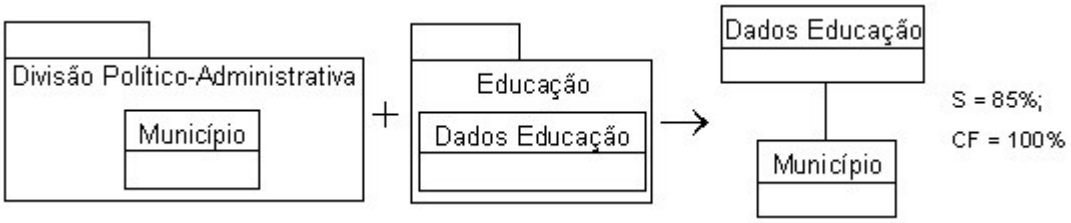


FIGURA 5.35 – Regra que substitui a apresentada na figura 5.34.

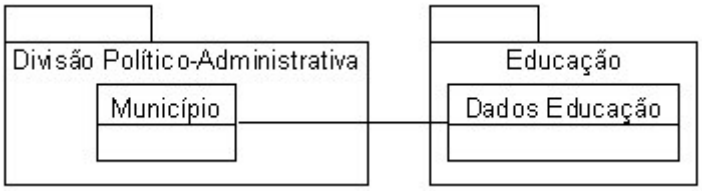


FIGURA 5.36 – Padrão identificado pela regra da figura 5.35.

A partir da identificação dos tipos de regras que não são de interesse, torna-se possível reduzir o número de regras de forma controlada e automática. Esse processo consiste, basicamente, em classificar cada regra de acordo com o tipo do subesquema presente em seu conseqüente e verificar se:

- estão presentes no antecedente da regra os elementos mínimos necessários à inferência de padrões do tipo que a regra identifica, conforme determina o requisito R2; e
- não existe mais itens e/ou elementos no antecedente da regra do que o mínimo necessário.

Com base nesses princípios, são definidos filtros usados para eliminar as regras que não são relevantes para identificação de padrões. Os filtros podem ser separados em dois grupos, os genéricos e os específicos por tipo de subesquema do conseqüente da regra. Filtros genéricos se aplicam a vários tipos de regras, enquanto os filtros específicos são definidos de acordo com o tipo de subesquema presente no conseqüente da regra. É importante ressaltar que essa estratégia parte do princípio de que toda regra é de interesse, a menos que seja descartada por algum dos filtros definidos. Após a aplicação dos filtros, algumas regras ainda podem ser descartadas pelo especialista humano.

A seguir são descritos os filtros identificados durante o desenvolvimento desse trabalho. Para cada filtro, são apresentadas a estrutura das regras por ele eliminadas assim como exemplos de tais regras, em notação UML.

É importante ressaltar que as medidas suporte e confiança não são utilizadas nesses filtros, pois já foram previamente aplicadas durante a MD.

5.2.1 Filtros Genéricos

a) Filtro F1

Elimina regras cujo conseqüente é um item do tipo "marca de esquema".

Esse tipo de regra não apresenta informação relevante, pois a "marca de esquema" só é utilizada na identificação de padrões formados por apenas uma classe ou apenas um pacote. A presença desse item só tem relevância no antecedente da regra.

Esse filtro elimina regras cuja estrutura é:

F1.ANTEC::= SUBESQ {(,"+", SUBESQ)};

F1.CONSEQ::= "E*";

Exemplo de regra eliminada pelo filtro é mostrado na figura 5.37.

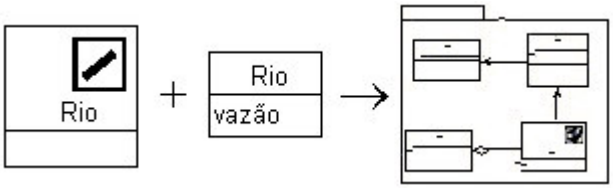


FIGURA 5.37 – Exemplo de regra eliminada pelo filtro F1.

b) Filtro F2

Elimina regras cujo antecedente possui o item "marca de esquema" e cujo item do conseqüente não é do tipo $Classe_{(1)}$ ou $Pacote_{(1)}$.

Uma vez que o item "marca de esquema" só utilizado em regras simples para identificar padrões formados por uma única instância de classe ou de pacote, qualquer outro tipo de subesquema presente no conseqüente da regra não representa informação relevante. A estrutura das regras descartadas por esse filtro é:

F2.ANTEC::= "E*";

F2.CONSEQ::= SUBESQ – $Classe_{(1)}$ – $Pacote_{(1)}$;

Exemplo de regra descartada por esse filtro é mostrado na figura 5.38.

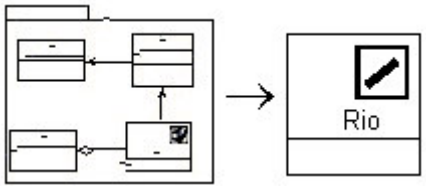


FIGURA 5.38 – Exemplo de regra eliminada pelo filtro F2.

c) Filtro F3

Elimina regras cujo antecedente possui o item "marca de esquema" acompanhado de outro(s) item(ns).

Para identificar candidatos a padrão formados apenas por uma instância de classe ou de pacote, não é necessário nenhum outro item no antecedente da regra além da "marca de esquema". A estrutura das regras descartadas por esse filtro é:

F3.ANTEC::= "E*" , "+", SUBESQ {(,"+", SUBESQ)}

F3.CONSEQ::= SUBESQ;

A figura 5.39 mostra um exemplo de regra descartada por esse filtro.

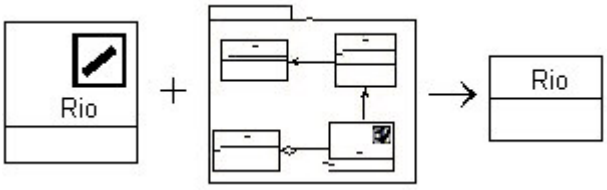


FIGURA 5.39 – Exemplo de regra eliminada pelo filtro F3.

d) Filtro F4

Elimina regras cujo subesquema do conseqüente não é do tipo Associação e cujo antecedente apresenta subesquema(s) desse tipo.

Item do tipo Associação é utilizado no antecedente da regra apenas para identificar padrões formados por mais de uma associação binária. Esse tipo de padrão, por sua vez, só é inferido por regras cujo conseqüente é subesquema do tipo Associação, conforme descrito na tabela 5.1. A estrutura das regras eliminadas por esse filtro é:

F4.ANTEC::= Associação {(,"+", ITEM)};

F4.CONSEQ::= ITEM – Associação;

A figura 5.40 mostra um exemplo de regra descartada por esse filtro.

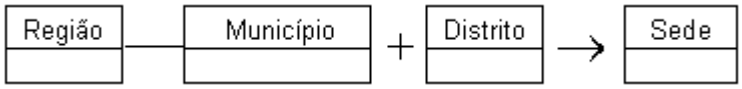


FIGURA 5.40 – Exemplo de regra eliminada pelo filtro F4.

e) Filtro F5

Elimina regras cujo subesquema do conseqüente não é do tipo Associação e seu antecedente, além de apresentar mais de um item, só possui subesquemas do tipo $Classe_{(1)}$.

Regras cujo antecedente possui apenas subesquema(s) do tipo $Classe_{(1)}$ são utilizadas para identificar padrões de dois tipos: aquele formado por uma classe com um atributo ou o composto por uma associação binária. Para identificar padrões formados por classe com atributo, é suficiente apenas um item no antecedente, correspondente à instância da classe do conseqüente. Padrões que possuem associação binária, por sua vez, são identificados através de regras cujo conseqüente é subesquema do tipo Associação.

A estrutura das regras eliminadas por esse filtro é mostrada a seguir. Exemplo desse tipo de regra é ilustrado na figura 5.41.

F5.ANTEC::= $Classe_{(1)}$, "+", $Classe_{(1)}$ {(,"+", $Classe_{(1)}$)};

F5.CONSEQ::= ITEM – Associação;

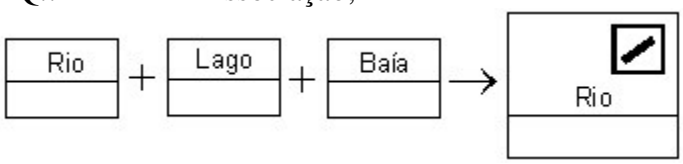


FIGURA 5.41 – Exemplo de regra eliminada pelo filtro F5.

f) Filtro F6

Descarta regras cujo antecedente possui apenas subesquemas do tipo Pacote$_{(1)}$.

Subesquemas desse tipo são considerados apenas para identificação de padrões formados por uma única instância de pacote. Nesse caso, esse tipo de item é apresentado no conseqüente da regra.

F6.ANTEC::= Pacote$_{(1)}$, {("+", Pacote $_{(1)}$)}

F6.CONSEQ::= ITEM

A figura 5.42 mostra um exemplo de regra descartada por esse filtro.

FIGURA 5.42 – Exemplo de regra eliminada pelo filtro F6.

5.2.2 Filtros específicos para conseqüente com subesquema do tipo Classe$_{(1)}$

a) Filtro F7

Elimina regras cujo antecedente não é formado exclusivamente pelo item "Marca de esquema".

Regras cujo conseqüente é subesquema do tipo Classe$_{(1)}$ são utilizadas para identificar candidatos a padrão formados apenas por uma classe, sendo necessário, para tanto, apenas a ocorrência do item "marca de esquema" no antecedente da regra.

F7.ANTEC::= "E*", "+", SUBESQ {(,"+", SUBESQ)};

F7.CONSEQ::= Classe$_{(1)}$;

A figura 5.43 mostra um exemplo de regra descartada por esse filtro.

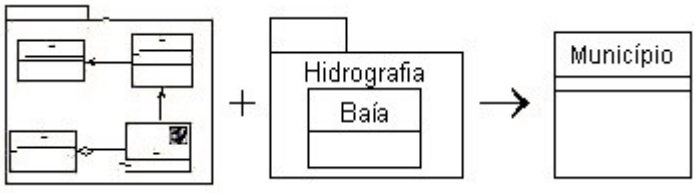


FIGURA 5.43 – Exemplo de regra eliminada pelo filtro F7.

5.2.3 Filtros específicos para conseqüente com subesquema do tipo Pacote$_{(1)}$

a) Filtro F8

Análogo ao filtro F7, elimina regras cujo antecedente não é formado exclusivamente pelo item "marca de esquema".

Regras cujo conseqüente é subesquema do tipo Pacote$_{(1)}$ são utilizadas para identificar candidatos a padrão formados apenas por uma instância de pacote. Para inferir esse tipo de padrão é necessária apenas a ocorrência do item "Marca de esquema" no antecedente da regra.

F8.ANTEC::= F7.ANTEC;

F8.CONSEQ::= Pacote$_{(1)}$;

Exemplo de regra descartada por esse filtro é mostrado na figura 5.44.

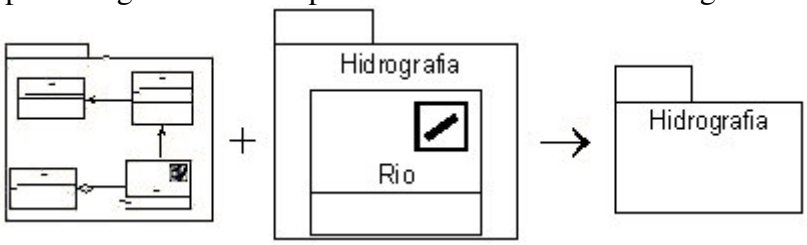


FIGURA 5.44 – Exemplo de regra eliminada pelo filtro F8.

5.2.4 Filtros específicos para conseqüente com subesquema do tipo Atributo$_{(1)}$-Classe

a) Filtro F9

São eliminadas regras cujos antecedentes não são formados apenas por subesquemas dos tipos Classe$_{(1)}$ e/ou Atributo$_{(1)}$-Classe. Além disso, são descartadas regras que possuem pelo menos uma classe no antecedente diferente da classe do conseqüente.

Regras com conseqüente do tipo Atributo$_{(1)}$-Classe são utilizadas para identificar candidatos a padrão formados por classe com atributo(s). Se a classe só possui um atributo, então é suficiente a existência, no antecedente, de um item com a mesma classe do conseqüente. Se a classe possui mais de um atributo, então são apresentados os outros atributos da mesma classe, no antecedente da regra.

F9.ANTEC::= ITEM - Classe - Atributo$_{(1)}$-Classe, {("+", ITEM - Classe - Atributo$_{(1)}$-Classe)};

F9.CONSEQ::= Atributo$_{(1)}$-Classe;

Exemplo de regra descartada por esse filtro é mostrado na figura 5.45.

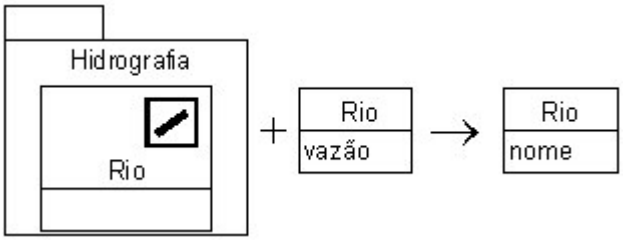


FIGURA 5.45 – Exemplo de regra eliminada pelo filtro F9.

b) Filtro F10

Elimina regras cujo antecedente apresenta redundância entre instâncias de classe.

Esse tipo de regra é gerado quando o candidato a padrão a ser identificado possui mais de um atributo. Além disso, o antecedente da regra é formado por subesquemas dos tipos Atributo$_{(1)}$-Classe e Classe$_{(1)}$. Nesse caso, o subesquema do tipo Classe$_{(1)}$ não agrega conhecimento e o mesmo padrão pode ser inferido a partir de uma regra sem esse item, a qual é mais simples e, portanto, com certeza também é gerada durante a MD.

A figura 5.46 ilustra um exemplo de regra descartada por esse filtro.

FIGURA 5.46 – Exemplo de regra eliminada pelo filtro F10.

5.2.5 Filtros específicos para conseqüente com subesquema do tipo Classe$_{(1)}$-Pacote.

a) Filtro F11

Elimina regras que não possuem no antecedente, pelo menos uma instância da classe e uma instância do pacote presentes no conseqüente da regra.

Esse filtro mantém apenas as regras cujo conseqüente é do tipo Classe$_{(1)}$-Pacote e que atendem ao requisito R2.

A figura 5.47 ilustra um exemplo de regra descartada por esse filtro.

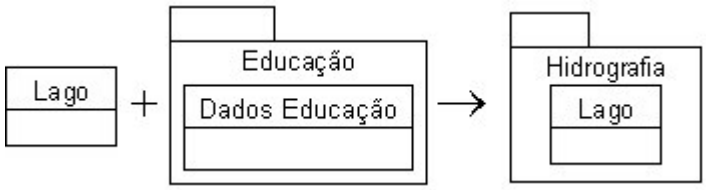


FIGURA 5.47 – Exemplo de regra eliminada pelo filtro F11.

b) Filtro F12

Descarta regras cujo antecedente possui item(ns) do tipo Atributo$_{(1)}$-Classe-Pacote.

Subesquemas desse tipo não são utilizados na inferência de candidatos a padrão formados por classes em pacote.

F12.ANTEC::= {(ITEM, "+")}, Atributo $_{(1)}$-Classe-Pacote;

F12.CONSEQ::= Classe$_{(1)}$-Pacote;

Exemplo de regra descartada por esse filtro é mostrada na figura 5.48.

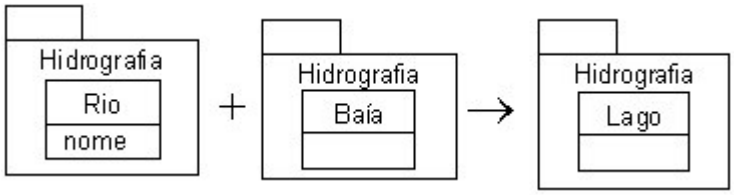


FIGURA 5.48 – Exemplo de regra eliminada pelo filtro F12.

c) Filtro F13

Elimina regras que apresentam, em seu antecedente, itens que possuem instância de pacote diferente da existente no conseqüente da regra.

Regras cujo conseqüente é subesquema do tipo Classe$_{(1)}$-Pacote são utilizadas para identificar padrões formados por uma ou mais instâncias de classe definidas em um mesmo pacote.

Exemplo de regra descartada por esse filtro é mostrada na figura 5.49.

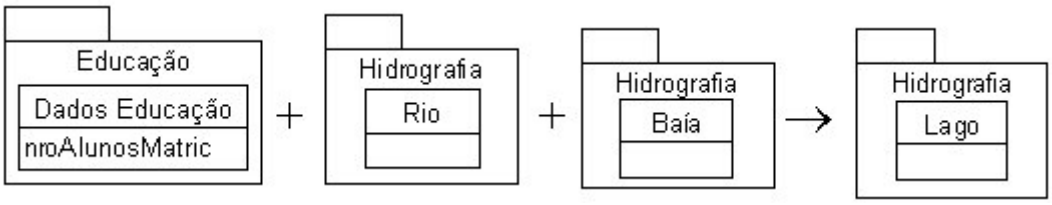


FIGURA 5.49 – Exemplo de regra eliminada pelo filtro F13.

5.2.6 Filtros específicos para conseqüente com subesquema do tipo Atributo$_{(1)}$-Classe-Pacote.

a) Filtro F14

Elimina regras cujo antecedente não é formado apenas por itens do tipo Classe$_{(1)}$-Pacote e/ou Atributo$_{(1)}$-Classe-Pacote.

Esse tipo de regra identifica construções freqüentes formadas por classe(s), com atributo(s), definidas em um único pacote. Se todas as classes do padrão estão definidas em pacote, o tipo de subesquema mais simples a ser encontrado na regra é Classe$_{(1)}$-Pacote. Subesquemas como Classe$_{(1)}$ ou Pacote$_{(1)}$, representam informação redundante.

F14.ANTEC::= {(ITEM - Classe$_{(1)}$-Pacote - Atributo$_{(1)}$-Classe-Pacote)};

F14.CONSEQ::= Atributo$_{(1)}$-Classe-Pacote;

A figura 5.50 ilustra um exemplo de regra descartada por esse filtro.

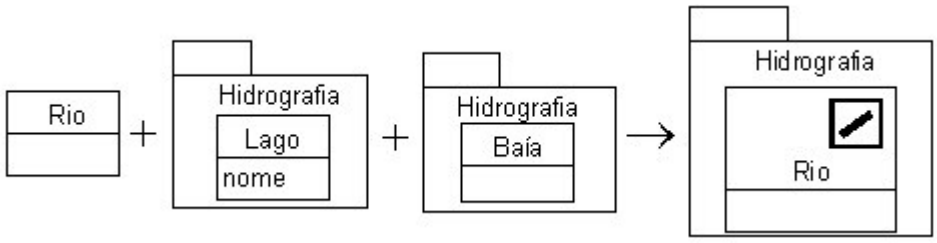


FIGURA 5.50 – Exemplo de regra eliminada pelo filtro F14.

b) Filtro F15

De modo análogo ao filtro F13, elimina regras que apresentam, em seu antecedente, itens com instância de pacote diferente da existente no conseqüente da regra.

Regras cujo conseqüente é subesquema do tipo Atributo$_{(1)}$-Classe-Pacote são utilizadas para identificar padrões formados por uma ou mais instâncias de classe definidas em um mesmo pacote.

Exemplo de regra descartada por esse filtro é mostrada na figura 5.51.

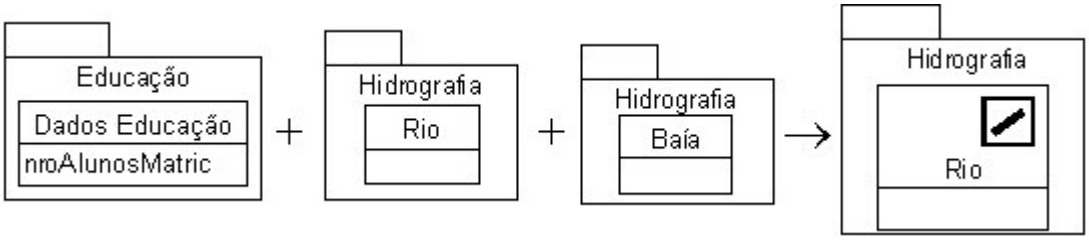


FIGURA 5.51 – Exemplo de regra eliminada pelo filtro F15.

5.2.7 Filtros específicos para conseqüente com subesquema do tipo Associação

a) Filtro F16

Se a associação do conseqüente não é um auto-relacionamento, elimina regras que não possuem no antecedente, no mínimo, as duas classes envolvidas no relacionamento do conseqüente.

Esse filtro elimina regras que não atendem ao requisito R2.

Exemplo de regra descartada por esse filtro é mostrada na figura 5.52.

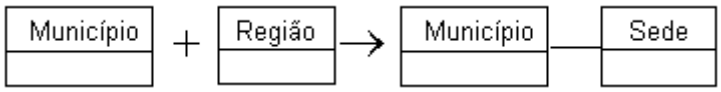


FIGURA 5.52 – Exemplo de regra eliminada pelo filtro F16.

b) Filtro F17

Elimina regras que apresentam no antecedente item do tipo $Classe_{(1)}$ com a instância da classe diferente daquelas apresentadas no conseqüente.

Instâncias de classe diferentes da apresentada no conseqüente só consistem em informação útil se estiverem representadas através de subesquemas dos tipos $Classe_{(1)}$-Pacote (determina o pacote no qual a classe está definida), $Atributo_{(1)}$-Classe (especifica um atributo da classe) ou $Atributo_{(1)}$-Classe-Pacote (combina as duas informações anteriores).

A figura 5.53 ilustra um exemplo de regra descartada por esse filtro.

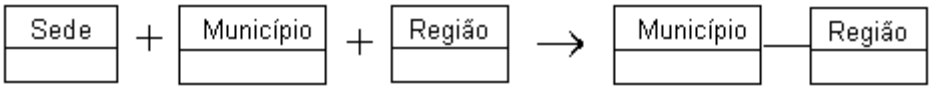


FIGURA 5.53 – Exemplo de regra eliminada pelo filtro F17.

c) Filtro F18

Elimina regras cujo antecedente apresenta item(ns) do tipo $Atributo_{(1)}$-Classe, $Classe_{(1)}$-Pacote e/ou $Atributo_{(1)}$-Classe-Pacote, se as instâncias de classe desse(s) item(ns) não forem iguais a nenhuma das classes presentes no conseqüente e a nenhuma classe de um relacionamento presente no antecedente da regra.

Esses tipos de subesquema apresentam informação adicional a respeito de classes envolvidas em associações binárias. Portanto, são significativos apenas quando representam instância de classe participante desse tipo de relacionamento.

A figura 5.54 ilustra um exemplo de regra descartada por esse filtro.

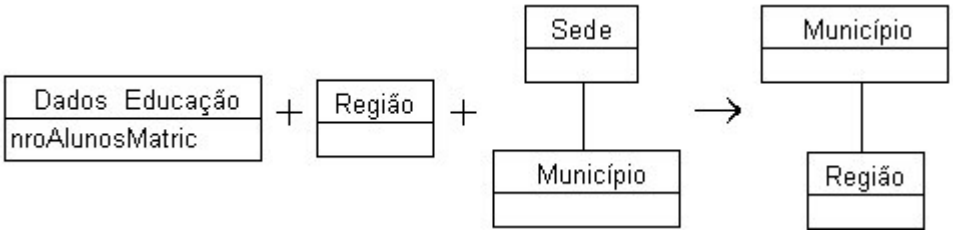


FIGURA 5.54 – Exemplo de regra eliminada pelo filtro F18.

6 Testes de Adequação

Para verificar a aplicabilidade do processo de DCBD na identificação de candidatos a padrão de análise para BDG é necessário observar se a técnica utilizada aponta construções freqüentes dentro de uma grande massa de dados. Para tanto, foram realizados testes em um ambiente controlado, onde as construções freqüentes são previamente conhecidas.

Os esquemas utilizados nos testes são fictícios e foram gerados automaticamente, pois:

- a utilização de esquemas reais de BDG é dificultada, uma vez que nem todos os aspectos referentes à etapa de preparação de dados foram tratados nesse trabalho;
- a criação de dados fictícios facilita o controle do processo;
- o objetivo dos testes consiste em verificar a adequação do processo de DCBD na solução do problema proposto (identificar candidatos a padrão de análise para BDG), não sendo objetivo do presente trabalho identificar verdadeiros padrões, os quais são comprovadamente utilizados durante a modelagem de sistemas reais.

O programa de geração de esquemas foi desenvolvido na linguagem Pascal, pois a mesma apresenta todos os operadores necessários à criação do algoritmo, além de ser de fácil utilização.

Para manter total controle sobre os esquemas e construções mais freqüentes geradas pelo algoritmo, alguns parâmetros devem ser informados. São eles:

- número total de esquemas a serem gerados;
- associações binárias freqüentes entre classes, assim como a freqüência (em %) de cada uma delas;
- atributos definidos em classes envolvidas em associações freqüentes e o pacote onde cada uma dessas classes está definida;
- quantidade máxima de classes que cada esquema gerado apresenta;
- quantidade máxima de atributos que podem ser definidos em cada classe; e
- o nome do arquivo de saída no qual devem ser armazenados os esquemas gerados.

A partir desses parâmetros, são gerados os esquemas. Para cada esquema, primeiramente, são armazenadas as classes envolvidas em associações freqüentes. As demais classes são criadas randomicamente até que seja atingido o número máximo de classes/esquema especificado pelo usuário.

Após armazenar todas as classes do esquema, o programa cria os subesquemas do tipo $Atributo_{(1)}$-Classe. Primeiramente são armazenados os atributos especificados pelo usuário e, em seguida, o algoritmo gera outros atributos. O número de atributos por classe é determinado aleatoriamente pelo algoritmo, mas é sempre menor ou igual ao número máximo de atributos/classe especificado pelo usuário.

A etapa seguinte consiste em armazenar os subesquemas dos tipos $Classe_{(1)}$-Pacote e $Atributo_{(1)}$-Classe-Pacote, de acordo com os pacotes onde cada classe está definida.

No último passo do algoritmo, são armazenadas as associações. Além das associações mais freqüentes especificadas pelo usuário, outras associações são geradas

aleatoriamente pelo algoritmo, de modo que cada classe está envolvida em pelo menos um relacionamento desse tipo.

Os esquemas gerados são decompostos, conforme especificado na seção 4.1.1 e armazenados e organizados como descrito na seção 4.2.2. Detalhes do algoritmo de geração dos esquemas são apresentados no Anexo 2.

Após a criação dos dados de entrada, o algoritmo de identificação de regras associativas disponível na ferramenta *IBM Intelligent Miner for Data* [IBM2002] é executado. Esta ferramenta foi utilizada em virtude de sua disponibilidade no PPGC-UFRGS e sua facilidade de utilização. Nessa etapa, os parâmetros são informados de modo a permitir a identificação de todas regras possíveis, não havendo limitação do número de itens que compõem seu antecedente. Essa ferramenta identifica regras cujos conseqüentes são compostos apenas por um item.

O arquivo de regras identificadas pela ferramenta é, então, pós-processado de forma automatizada, através de um algoritmo implementado para automatizar a aplicação dos filtros apresentados na seção 5.1.2. O algoritmo que automatiza o pós-processamento das regras está disponível no Anexo 3.

As regras não eliminadas pelos filtros são analisadas manualmente, a fim de verificar se apresentam a descrição das construções freqüentes existentes no arquivo de entrada.

O procedimento acima apresentado foi realizado três vezes. Para cada um dos estudos de caso foram gerados 9000 esquemas com 15 classes cada um. Os estudos de casos apresentados nas seções a seguir diferem entre si pelas construções mais freqüentes especificadas pelo usuário.

6.1 Estudo de Caso 1

Na primeira instanciação do processo, foram gerados 9000 esquemas, compostos por 15 classes, cada um. A construção da figura 6.1 está presente em 70% dos esquemas.

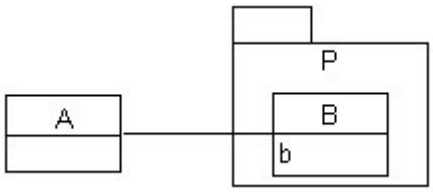


FIGURA 6.1 – Construção freqüente do estudo de caso 1.

A ferramenta *Intelligent Miner for Data* foi utilizada, com suporte especificado em 65%. Durante a execução, 1016 regras foram geradas.

Sobre o arquivo de regras resultante, foi executado o algoritmo de pós-processamento. O número total de regras após a aplicação dos filtros é 13, dentre as quais 10 identificam algum tipo de padrão implícito na construção freqüente apresentada na entrada do algoritmo de MD. A regra que apresenta o padrão completo é do tipo C7 e pode ser vista na figura 6.2.

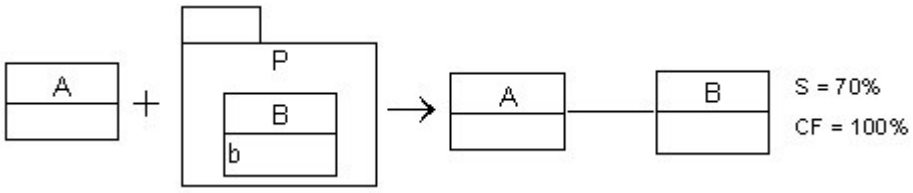


FIGURA 6.2 – Regra que identifica a construção da figura 6.1.

6.2 Estudo de Caso 2

Na segunda execução do processo, foram gerados 9000 esquemas compostos por 15 classes, cada um. A construção da figura 6.3 está presente em 70% dos esquemas, enquanto a da figura 6.4 encontra-se em 80%.

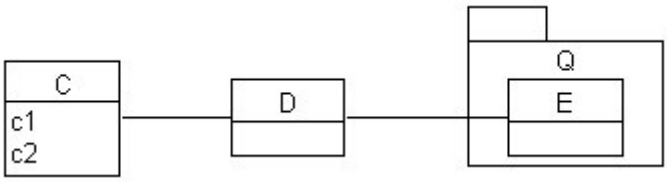


FIGURA 6.3 – Construção com freqüência de 70% no estudo de caso 2.

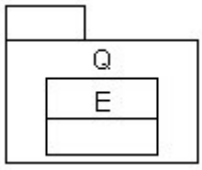


FIGURA 6.4 – Construção com freqüência de 80% no estudo de caso 2.

O suporte mínimo especificado na ferramenta *Intelligent Miner for Data* foi de 65%. Ao final da execução 5110 regras foram apresentadas.

A execução do algoritmo de pós-processamento sobre o arquivo resultante da MD resultou em um total de 53 regras, das quais 21 podem ser interpretadas como candidato a padrão de análise. Dentre as regras remanescentes, destacam-se as mostradas nas figuras 6.5 (regra complexa do tipo C10c) e 6.6 (regra simples do tipo S4), as quais indicam os padrões mostrados, respectivamente, nas figuras 6.3 e 6.4.

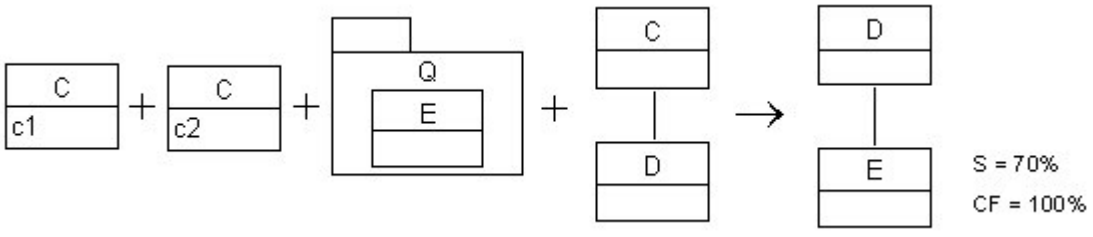


FIGURA 6.5 – Regra que identifica a construção da figura 6.3.

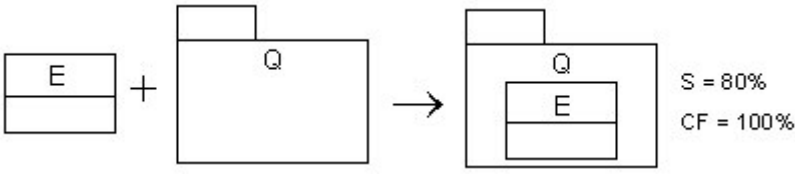


FIGURA 6.6 – Regra que identifica a construção da figura 6.4.

6.3 Estudo de Caso 3

Para o terceiro estudo de caso, novamente, foram gerados 9000 esquemas compostos por 15 classes, cada um. As construções mostradas nas figuras 6.7 e 6.8 estão presentes no arquivo de entrada em 60% e 70% das transações, respectivamente.

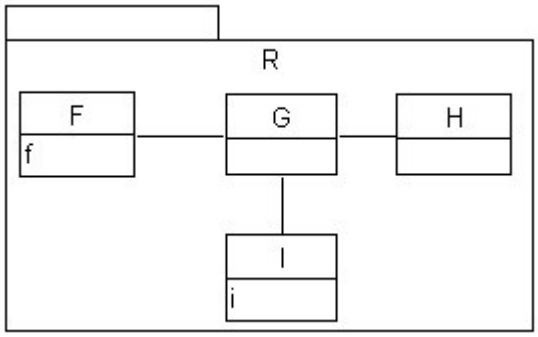


FIGURA 6.7 – Construção com ocorrência de 60% no estudo de caso 3.

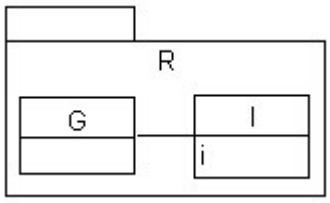


FIGURA 6.8 – Construção com 70% de ocorrência no estudo de caso 3.

A mineração de dados foi executada três vezes, utilizando o mesmo arquivo de entrada. Na primeira execução, o suporte mínimo foi especificado em 55%. A mineração de dados resultou em mais de 1 milhão de regras, as quais foram reduzidas para pouco mais 11 mil, após a aplicação dos filtros no pós-processamento. As figuras 6.9 e 6.10 mostram exemplos de regras interessantes, mantidas após o pós-processamento dos dados. A regra da figura 6.9 é do tipo C10c, enquanto a da 6.10 é C7.

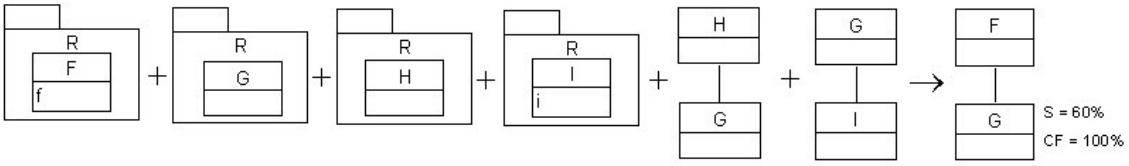


FIGURA 6.9 – Regra que identifica a construção da figura 6.8.

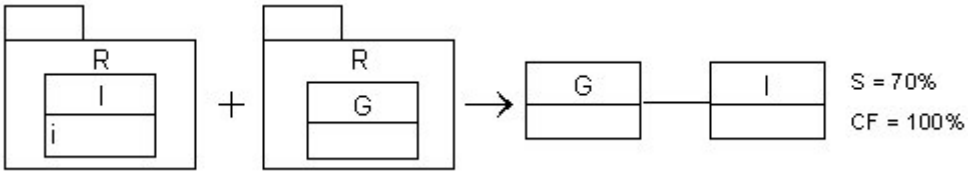


FIGURA 6.10 – Regra que identifica a construção da figura 6.7.

A segunda execução do algoritmo de MD foi realizada com suporte especificado em 65%. Ao final do processamento, 2295 regras foram apresentadas. Esse número foi reduzido para 12, após a execução do algoritmo de pós-processamento. Dentre as regras

remanescentes, 11 possuem significância no contexto da aplicação, dentre as quais está aquela apresentada na figura 6.9.

A terceira mineração de dados foi realizada com suporte especificado em 75%. Nenhuma regra foi gerada como resultado, devido ao alto valor do suporte mínimo especificado.

7 Conclusão

O emprego de padrões de análise tem sido apresentado como alternativa no sentido de estimular e facilitar a modelagem conceitual de BDG, pois simplifica a criação e atualização de esquemas conceituais. A fim de facilitar tanto a disponibilização quanto a recuperação dessas construções, padrões são organizados em catálogos.

A criação de um catálogo de padrões leva em consideração três aspectos principais: aquisição de conhecimento, atualização e consulta. O problema abordado nesse trabalho consiste na aquisição de conhecimento, uma vez que, segundo Hennigen [HEN97] as demais questões são comumente objetivo de estudo. Além disso, o processo de identificação de padrões tem se mostrado um entrave na popularização do uso dessas construções.

As técnicas atualmente utilizadas com o objetivo de identificar padrões de análise são centradas no conhecimento de especialistas, tornando o processo lento e subjetivo. A subjetividade é especialmente prejudicial, pois possibilita que os padrões propostos por um especialista sejam refutados por outros com diferentes experiências de projeto. Se por um lado essas metodologias dificultam a identificação de novos padrões, por outro, elas possibilitam que sejam propostos padrões de forma descontrolada, conforme afirma Kerth em [FRA95].

Nesse contexto, o presente trabalho propôs a aplicação do processo de DCBD como alternativa às metodologias de identificação de candidatos a padrão de análise baseadas exclusivamente no conhecimento de especialista, como as citadas por Bushman [BUS96] e Rising [RIS99]. Esta proposta, todavia, também pode ser utilizada como ferramenta de apoio ao especialista que segue tais metodologias. A partir de esquemas conceituais de BDG, técnicas de mineração de dados são aplicadas com o objetivo de identificar as construções freqüentemente utilizadas na modelagem da realidade. O pós-processamento dos resultados viabiliza a análise das regras resultantes por parte de um especialista, o qual confirma ou não o candidato inferido como padrão de análise para BDG.

O processo de DCBD foi utilizado, pois tem como objetivo automatizar (ou pelo menos, semi-automatizar) a análise de grandes volumes de dados. Desse modo, sua aplicação possibilita a análise de um número muito maior de esquemas em relação ao que é realizado pelos métodos tradicionalmente empregados, sendo possível comparar esquemas de diferentes especialistas.

O trabalho apresenta a aplicação da técnica de identificação de regras associativas como alternativa à análise realizada pelo especialista a fim de identificar candidatos a padrão de análise para BDG. A escolha dessa técnica leva em consideração os critérios citados na bibliografia consultada, dentre os quais destacam-se a sua adequação ao objetivo da aplicação do processo e o formato de representação do resultado.

A fim de possibilitar a mineração, investigou-se como devem ser conduzidas as atividades referentes à preparação dos esquemas conceituais de BDG a serem minerados. No que se refere a esse aspecto, maior ênfase é dada às atividades de decomposição de esquemas e armazenamento em formato compatível com aquele utilizado pela técnica de MD selecionada.

Além disso, foi realizado um estudo a respeito dos resultados obtidos durante a mineração, quais sejam:

- todas as regras geradas são formadas por construções freqüentes;

- cada padrão está implícito em mais de uma regra;
- um subconjunto das regras geradas é suficiente para identificar todos os candidatos a padrão existentes nos esquemas minerados; e
- algumas regras estão contidas em outras.

Com base no conhecimento obtido, foi possível identificar um subconjunto de regras de interesse e um subconjunto de regras que, apesar de serem formadas por construções freqüentes, são redundantes ou não agregam informação útil para a identificação dos padrões de análise buscados. Esse levantamento possibilitou a criação de um programa de computador que elimina, de forma automática, regras que não agregam informação útil ao objetivo final do processo.

Por fim, a execução de testes mostra que, de fato, a utilização da técnica de identificação de regras associativas é um grande passo no sentido de agilizar a detecção de novos padrões de análise. O algoritmo de pós-processamento também se mostrou útil no sentido de diminuir a quantidade de regras identificadas, facilitando a análise manual a ser realizada pelo especialista. Além disso, a estratégia de decomposição de esquemas, que considera apenas alguns poucos tipos de subesquema, mostrou-se adequada às necessidades da aplicação.

Durante os testes também foi possível detectar que, para a identificação de padrões de análise para BDG, não é apenas o valor do suporte que determina a quantidade de regras geradas. Nesse caso, além do valor do suporte mínimo, especificado pelo usuário, a complexidade do(s) padrão(ões) implícito(s) nos esquemas analisados também tem influência sobre a quantidade de regras geradas. Essa constatação foi possível através de resultados de testes, onde foi possível observar que a variação da construção freqüente e do número de itens nas regras alteram a quantidade de regras geradas pelo algoritmo de MD. Obviamente, se for especificado um valor de suporte mínimo maior do que o número de ocorrências da construção, a quantidade de regras também é reduzida.

Vale ressaltar ainda que o estudo apresentado pode ser aplicado em diversas outras áreas e não apenas em SIG.

Embora os resultados da pesquisa tenham sido satisfatórios, alguns aspectos ainda devem ser considerados em trabalhos futuros, dentre os quais destacam-se:

- outros construtores do modelo de dados devem ser considerados;
- devem ser definidos critérios para seleção de esquemas a minerar;
- problemas de sinônimos, antônimos e parônimos devem ser tratados nos dados de entrada;
- testes, utilizando esquemas de aplicações reais, devem ser realizados;
- novos filtros devem ser investigados e implementados, a fim de que um número maior de regras seja descartado de forma automática; e
- candidatos a padrão inferidos a partir de regras já identificadas como de interesse devem poder ser apresentados em um formato mais inteligível pelo especialista como na forma de diagrama de classe da UML.

Do ponto de vista dos algoritmos de MD, trabalhos futuros podem:

- verificar os resultados gerados por outras técnicas de MD e compará-los aos resultados obtidos neste trabalho;
- investigar a utilização de taxonomias, com o objetivo de serem identificadas menos regras; e
- utilizar ou implementar um algoritmo de identificação de regras associativas que mantenha no resultado apenas aquelas mais abrangentes, eliminando

regras contidas em outras desde que seus valores de suporte e confiança sejam iguais aos da regra mais completa. Essa medida também deve diminuir a quantidade de regras geradas durante etapa de MD.

Referências

[AGE98] AGERBO, E.; CORNILS, A. How to Preserve the Benefits of Design Patterns. **SIGPLAN Notices**, New York, v.33, n.10, p.134-143, Oct. 1998.

[AGR94] AGRAWAL, R.; SRIKANT, R. Fast Algorithms for Mining Association Rules in Large Databases. In: VLDB, 1994. **Proceedings…**. Hove: Morgan Kaufmann, 1994.

[AGR93] AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Mining Association Rules Between sets of Items in Large Databases. In: INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1993, Washignton. **Proceedings…** [S.l.: s.n.], 1993.

[ALE77] ALEXANDER, C. et al. **A Pattern Language:** Towns, Buildings, Construction. England: Oxford University Press, 1977.

[APP97] APPLETON, B. Patterns and Software: Essential Concept and Terminology. **Object Magazine Online**, v.3, n.5, May 1997. Disponível em: <http://www.enteract.com/~bradapp/docs/patterns-intro.html>. Acesso em: 19 nov.2001.

[ARO89] ARONOFF, S. **Geographic Information Systems.** Canada: WDL Publications, 1989.

[BAS2002] BASSALO, G. H. M.; IOCHPE, C.; BIGOLIN, N. M. Representando Esquemas de Bancos de dados Geográficos no Formato Atributo-Valor para a Inferência de Padrões de Análise. In: SIMPÓSIO BRASILEIRO DE GEOINFORMÁTICA, 4., 2002. **Anais...** Belo Horizonte: SBC, 2002.

[BAS2001] BASSALO, G. H. M. **Integração de modelos conceituais para sistemas de informação geográfica voltada à preparação de esquemas de bancos de dados geográficos para utilização em ferramentas de descoberta de conhecimento**. 2001. 64f. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[BAN98] BANSIYA, J. Automating Design-Pattern Identification. **Dr. Dobb's Journal** June 1998. Disponível em: <http://www.ddj.com/documents/s=919/ddj9806a/9806a.htm>. Acesso em: 26 nov.2002.

[BER97] BERRY, M.; LINOFF, G. **Data Mining Techniques:** for marketing, sales, and customer support. New York: John Wiley & Sons Inc, 1997.

[BLU82] BLUM, R. L. Discovery, Confirmation, and Incorporation of Causal Relationships from Large Time-Oriented Clinical Database: The RX Project. **Computers and Biomedical Research**, [S.l.], v.15, p.164-187, 1982.

[BOO99] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Reading, Massachusetts: Addison-Wesley, 1999.

[BOR97] BORGES, K.A.V. **Modelagem de Dados Geográficos:** uma Extensão do Modelo OMT para Aplicações Geográficas. 1997. Dissertação (Mestrado em) - Escola de Governo de MG/FJP, Belo Horizonte.

[BRA98] BRAND, E.; GERRITSEN, R. Data Mining and Knowledge Discovery. **DBMS**, [S.l.], v.11, n.9, Aug. 1998. Disponível em: <http://www.dbmsmag.com/9807m01.html>. Acesso em: 01 nov. 2001.

[BRA98a] BRAND, E.; GERRITSEN, R. Classification and Regression. **DBMS**, [S.l.], v.11, n.9, Aug. 1998. Disponível em: <http://www.dbmsmag.com/9807m04.html>. Acesso em: 01 nov. 2001.

[BRA98b] BRAND, E.; GERRITSEN, R. Association and Sequencing. **DBMS**, [S.l.], v.11, n.9, Aug. 1998. Disponível em: <http://www.dbmsmag.com/9807m03.html>. Acesso em: 01 nov. 2001.

[BRI95] BRITISH COLUMBIA. Geographic Data. Ministry of Sustainable Resource Management. **SAIF 3.2 Specification**. 1995-2001. Disponível em: <ftp://home.gdbc.gov.bc.ca/SAIF/>. Acesso em: set. 2001.

[BRO2001] BROWN, K. **Design Reverse-Engineering and Automated Design Pattern detection in Smalltalk**. Disponível em: <http://www2.ncsu.edu/eos/info/tasug/kbrown/thesis2.htm>. Acesso em: 15 mar. 2001.

[BUR 97] BURROUGH, P. A.; MCDONNELL, R. A. **Principles of Geographical Information Systems**. Great Britain: Oxford university Press, 1997.

[BUR87] BURTON, B. A. et al. The Reusable Software Library. **IEEE Software**, Los Alamitos, v.4, n.4, p.25-33, 1987.

[BUS96] BUSCHMANN, F. **Pattern-oriented software arquitecture :** a system of patterns. Chichester : John Wiley, 1996.

[CAL91] CALDIERA, G.; BASILI, V. R. Identifying and Qualifying Reusable Software Componentes. **Computer**, Los Alamitos, v.24, n.2, p.61-70, Feb. 1991.

[CHE96] CHEN, M.; HAN, J.; YU, P. S. Data Mining: An Overview from Database Perspective. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 6, n. 8, p. 866-883, 1996.

[COA97] COAD, P. **Object Models:** Strategies, Patterns and Aplications. 2nd ed. New Jersey: Yourdon Press, 1997.

[COA96] COAD, P.; YOURDON, E. **Análise baseada em objetos**. Rio de Janeiro: Campus, 1996.

[COP95] COPLIEN, J.; SCHMIDT, D. C. **Pattern Languages of Program Design**. Massachusetts: Addison-Wesley, 1995.

[COP92] COPLIEN, J. **Advanced C++**: Programming Styles and Idioms. Massachusetts: Addison-Wesley, 1992.

[CYB2001] CYBULSKI , J. L. **Application of Software Reuse Methods to Requirements Elicitation from Informal Requirements Texts.** 2001. PhD Thesis. School of Computer Science and Computer Engineering, La Trobe University. Disponível em: <http://www.dis.unimelb.edu.au/staff/jacob/>. Acesso em: 28 fev. 2002.

[DEV91] DEVANBU, P. et al. LaSSIE: A knowledge-based software information system. **Communications of the ACM**, New York, v.34, n.5, p.34-49, May 1991.

[ELM2000] ELMASRI, R; NAVATHE, S. B. **Fundamentals of database systems**. Reading: Addison-Wesley, 2000.

[FAY 96] FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From Data Mining to Knowledge Discovery in Databases. **AI Magazine**, Menlo Park, v.17, n.3, p.37-54, Fall 1996.

[FAY96a] FAYYAD, U. M.; PIATETSKY-SHAPIRO, G.; SMYTH, P. The KDD Process for Extracting Useful Knowledge from Volumes of Data. **Communications of the ACM**, New York, v.39, n.11, Nov. 1996.

[FER2000] FERNADEZ, E. B.; YUAN, X. Semantic Analysis Patterns. In: INTERNATIONAL CONFERENCE ON CONCEPTUAL MODELING, ER, 19., 2000, Salt Lake City. **Conceptual Modeling**: proceedings. Berlin: Springer-Verlag, 2000. p. 183-195.

[FER98] FERNADEZ, E. B. Building Systems Using Analysis Patterns. In: INTERNATIONAL WORKSHOP ON SOFTWARE ARCHITECTURE, 3., 1998. **Proceedings...**. New York: ACM Press, 1998.

[FER99] FERREIRA; A. B. H. **Novo Aurélio Século XXI:** o Dicionário da Língua Portuguesa. Rio de Janeiro: Nova Fronteira, 1999.

[FOW97] FOWLER, M. **Analysis Patterns:** reusable object models. Menlo Park, CA: Addison Wesley Longman, 1997.

[FOW96] FOWLER, M. A Survey of Object-Oriented Analysis and Design Methods. In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING, 10., 1996, Linz, Austria. **Object-Oriented Programming:** proceedings. Berlin: Springer-Verlag, 996.

[FRA95] FRASER, S. et al. Patterns: cult to culture? **SIGPLAN Notices**, New York, v. 30, n.10, p.134-143, Oct. 1995.

[FUR98] FURLAN, J. D. **Modelagem de Objetos através da UML – the Unified Modeling Language**. São Paulo: Makron Books, 1998.

[GAM2000] GAMMA, E. et al. **Padrões de Projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000.

[GAM95] GAMMA, E. et al. **Design patterns**: elements of reusable object-oriented software. Reading, Massachusetts: Addison-Wesley, 1995.

[GOE99] GOEBEL, M.; GRUENWALD, L. A Survey of Data Mining and Knowledge Discovery Software Tools. **SIGKDD Explorations**, [S.l.], v.1, n.1, p.20-32, June/1999.

[GOO92] GOODCHILD, M. F. Geographical Data Modeling. **Computers & Geosciences**, London, v.18, n.4, p.401-408, 1992.

[HAN2001] HAN, J.; KAMBER, M. **Data Mining**: concepts and techniques. San Diego, CA: Morgan Kaufmann, 2001.

[HAN95] HAN, J.; FU, Y. Discovery of Multiple-Level Association Rules from Large Databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASE, VLDB, 21., 1995, Zürich, Switzerland. **Proceedings…** San Francisco: Morgan Kaufmann, 1995. p. 420-431.

[HAY95] HAY, D. C. **Data Model Patterns:** conventions of thought. New York: Dorset House Publishing, 1995.

[HEN97] HENNINGER, S. An Evolutionary Approach to Constructing Effective Software Reuse Repositories. **ACM Transactions on Software Engineering Methodology**, New York, v.2, n.6, p. 111 – 140,

[IBM2002] IBM. **DB2 Intelligent Miner for Data**. Disponível em: <http://www.ibm.com>. Acesso em: 11 abr. 2002.

[JOH99] JOHANNESSON, P.; WOHED, P. The Deontic Pattern – a framework for domain analysis in information systems design. **Data & Knowledge Engineering**, [S.l.], v.31, 1999.

[JOH98] JOHANNESSON, P.; WOHED, P. Deontic Analysis Patterns. In: THE INTERNATIONAL WORKSHOP ON THE LANGUAGE ACTION PERSPECTIVE ON COMMUNICATION MODELLING, LAP, 1998. **Proceedings...** Disponível em: <http://www.dsv.su.se/~petia/papers/lap98.pdf>. Acesso em: 1 jul.2002.

[KÖS97] KÖSTERS, G.; PAGEL, B.; SIX, H. GIS-application development with GeoOOA. **International Journal of Geographical Information Science,** London, v.11, n.4, 1997.

[LAR2000] LARGMAN, C. **Utilizando UML e Padrões**: uma introdução a análise e ao projeto orientados a objetos. Porto Alegre: Bookman, 2000.

[LIS2001] LISBOA FILHO, J.; IOCHPE, C.; BORGES, K. A. V. Padrões de Análise para reutilização de Esquemas de Dados de SIG em Aplicações de Gestão Urbana. In: CONFERÊNCIA LATINOAMERICANA DE INFORMÁTICA, CLEI, 27., 2001. **[Artículos].** Mérida: Universidad de Los Angeles, 2001.

[LIS2000] LISBOA FILHO, J. **Projeto conceitual de banco de dados geográficos através da reutilização de esquemas, utilizando padrões de analise e um framework conceitual**. 2000. Tese (Doutorado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[LIS99] LISBOA FILHO, J.; IOCHPE, C. Um Estudo sobre Modelos Conceituais de Dados para Projeto de Bancos de Dados Geográficos. **Revista IP – Informática Pública**, Belo Horizonte, MG, v.1, n.2, 1999.

[LIS98] LISBOA FILHO, J.; IOCHPE, C.; BEARD, K. Applying Analysis Patterns in the GIS Domain. In: ANNUAL COLLOQUIUM OF THE SPATIAL INFORMATION RESEARCH CENTRE, SIRC, 1998. **Proceedings…**. Dunedin: SIRC, University of Otago, 1998.

[MES2001] MESZAROS, G.; DOBLE, J. **A Pattern Language for Pattern Writing**. Disponível em: <http://hillside.net/patterns/writing/patternwritingpaper.htm>. Acesso em: 19 nov. 2001.

[MIC2000] MICHAIL, A. Data Mining Library Reuse Patterns using Generalized Association Rules. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 22., 2000, Limerick Ireland. **Proceedings...**.New York: ACM, 2000. p.167-176.

[PAR97] PARENT, C.; SPACCAPIETRA, S.; ZIMANYI, E. Conceptual Modeling for Federated GIS over the Web. INTERNATIONAL SYMPOSIUM ON INFORMATION SYSTEMS AND TECHNOLOGY FOR NETWORK SOCIETY, 1997, Fukuoka, Japan. **Proceedings...** [S.l.: s.n.], 1997. p. 173-182.

[PEU94] PEUQUET, D. It's about time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems. In: **Annuals of the Association of American Geographers**. [S.l.], 1994.

[PIA91] PIATETSKY-SHAPIRO, G. Knowledge Discovery in Real Databases: A workshop report. **AI Magazine**, Menlo Park, v. 11, n. 5, Jan 1991. Disponível em: <http://www.channel1.com/users/gps1/gpspubs/aimag-kdd-1991-report.pdf>. Acesso em: ago. 2002.

[RAM 94] RAMIREZ, M. R. **Sistemas Gerenciadores de Bancos de Dados para Geoprocessamento.** 1994. Dissertação (Mestrado) - COPPE/UFRJ, Rio de Janeiro.

[REF2000] REFACTORY INC, THE; HILLSIDE GROUP, THE. **Patterns Home Page**. 2000-2003. Disponível em: <http://hillside.net/patterns/>. Acesso em: 22 feb. 2003.

[RIE96] RIEHLE, D; ZÜLLIGHOVEN, H. Understanding and Using Patterns in Software Development. **Theory and Practice of Object Systems**, New York, v.2, n.1, p.3-13, 1996.

[RIS99] RISING, L. Patterns Mining. In: ZAMIR, S. (Ed.). **Handbook of Object Technology.** [S.l.]: CRC Press, 1999. Disponível em: <http://www.agcs.com/supportv2/techpapers/patterns/papers/mining.htm> Acesso em: ago.2002.

[ROB93] ROBERTSON, S.; STRUNCH, K. Reusing the products of analysis. In: INTERNATIONAL WORKSHOP ON SOFTWARE REUSABILITY, 1993, Lucca, Italy. **Proceedings...** [S.l.: s.n.], 1993.

[SAV98] SAVASARE, A.; OMIECINSKI, E.; NAVATHE, S. Mining for Strong Negative Associations in a Large Database Customer Transactions. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 14., 1998. **Proceedings…** [S.l.: s.n.], 1998. p.494-502.

[SHU96] SHULL, F.; MELO, W. L.; BASILI, V. R. **An Inductive Method For Discovering Design Patterns From Object-Oriented Software Systems**. 1996. University of Maryland, Computer Science Department, College Park, MD, 20742 USA (Technical Report). Disponível em: <http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/CS-TR-3597.pdf>. Acesso em: set.2002.

[SIL2002] SILVA, C. M. S.; IOCHPE, C.; ENGEL, P. M. Applying the process of knowledge discovery in databases to identify analysis patterns for reuse in geographic database design. In: BRAZILIAN SYMPOSIUM ON ARTIFICIAL INTELLIGENCE, 16., 2002, Porto de Galinhas. **Advances in artificial intelligence**: proceedings. Berlin : Springer-Verlag, c2002. p. 291-301.

[SIL2001] SILVA, C. M. S. **Estudo de Técnicas para Suporte à Geração de Catálogos de Padrões de Análise para Projeto de Bancos de Dados Geográficos**. 2001. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

[SRI96] SRIKANT, R.; AGRAWAL, R. Mining Quantitative Association Rules in Large Relational Tables. **SIGMOD Conference**. Montreal: ACMPress,1996. p.1-12.

[SRI95] SRIKANT, R.; AGRAWAL, R. Mining Generalized Association Rules. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASE, VLDB, 21., 1995, Zürich, Switzerland. **Proceedings…** San Francisco: Morgan Kaufmann, 1995.

[TEI 97] TEIXEIRA, A.L.A.; CHRISTOFOLETTI, A. **Sistemas de Informação Geográfica –Dicionário Ilustrado**. [S.l.]: Ed. Hucitec, 1997. Disponível em: <http://www.fatorgis.com.br/aplicativos/dicionario.cgi>. Acesso em: 01 mar. 2001.

[WAN2000] WANG, W.; YANG, J; YU, P. S. Efficient mining of weighted association rules (WAR). In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, KDD, 6., 2000, Boston. **Proceedings…**[S.l.: s.n.], 2000. p. 270-274.

[WEB2000] WEBB, G. I. Efficient search for association rules. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, KDD, 6., 2000, Boston. **Proceedings…**[S.l.: s.n.], 2000. p.99-107

[WIL88] WILLET, P. Recent trends in hierarchic document clustering: a critical review. **Information Processing & Management**, Oxford, v.24, n.5, p.577-597, 1998.

[WIT2000] WITTEN, I. H.; FRANK, E. **Data mining**: practical machine learning tools and techniques with Java implementations. San Francisco: Morgan Kaufmann, 2000.

[ZAK2000] ZAKI, M. J. Generating Non-redudant Association Rules. In: ACM SIGKDD INTERNATIONAL CONFERENCE ON KNOWLEDGE DISCOVERY AND DATA MINING, KDD, 6., 2000, Boston. **Proceedings…**[S.l.: s.n.], 2000. p.34-43.

Glossário

Agrupamento: técnicas de agrupamento geram modelos descritivos que buscam agrupar objetos em classes de objetos similares. Os dados de entrada são analisados de modo a identificar classes naturais (ou grupos).

Análise de Associações: busca encontrar relacionamentos interessantes entre itens de um grande conjunto de dados de um domínio específico. Os relacionamentos identificados representam padrões de comportamento.

Aprendizado em Lote: considera todo o conjunto de dados de uma única vez. A modificação do arquivo de entrada implica na necessidade de execução da técnica de MD novamente.

Aprendizado Incremental: não desconsidera resultados de minerações anteriores quando o arquivo de entrada é modificado.

Aprendizado Não-supervisionado: utilizado para descrição e busca identificar como os dados estão relacionados, quais itens são similares, quais são diferentes e de que forma. Nesse caso, nenhuma classe é predefinida, cabendo ao algoritmo manipular os dados de modo a determiná-las.

Aprendizado Supervisionado: é baseado em exemplos e, geralmente, é utilizado por técnicas de MD que geram modelos preditivos. Utiliza dados de entrada previamente classificados por um especialista no domínio da aplicação.

Caixa Preta (Representação do conhecimento do tipo): não apresentam a estrutura do padrão de modo explícito.

Caixa Transparente (Representação do conhecimento do tipo): representam os padrões em termos de uma estrutura que pode ser examinada, sobre a qual é possível raciocinar e que pode ser usada para guiar decisões futuras.

Candidato a Padrão de Análise: construção freqüente, ainda não validada, dado um conjunto de esquemas de BDG.

Classificação: tarefa realizada pela MD que consiste em aprender uma função que mapeia (classifica) um item para uma dentre várias classes pré-definidas.

Confiança (da regra): é a relação entre o número de transações que contém todos os itens da regra (A U C) e o número de transações que contém, pelo menos, os itens do seu antecedente (A).

Conhecimento Objetivo do Especialista: domínio dos construtores dos modelos de dados e das regras de combinação dos mesmos.

Conhecimento Subjetivo do Especialista: experiência adquirida durante a concepção de esquemas de banco de dados.

Construtor: conceito definido em um modelo de dados utilizado para modelar a realidade.

Construtor Fraco: construtor do modelo que não pode ser definido isoladamente no esquema, pois individualmente não formam subesquemas significativos. Sua definição dependente do seu relacionamento com um construtor forte, como ocorre com atributos e associações.

Construtor Forte: é um construtor do modelo que, isoladamente, constitui um subesquema significativo, como pacote e classe na UML.

Elemento: instância de construtor utilizada em um esquema específico.

Elemento Forte: instância de construtor forte do modelo.

Elemento Fraco: instância de construtor fraco do modelo.

Large itemset: combinação de itens do arquivo de entrada que possue suporte acima do limite especificado.

Marca de esquema: item criado com o objetivo de registrar a ocorrência de um esquema qualquer, não representando nenhum tipo de subesquema gerado a partir da decomposição. Regras cujo antecedente é formado apenas por esse item indicam que a criação de esquemas implica na utilização da construção presente no seu conseqüente.

Modelo Descritivo: tipo de modelo gerado por técnicas de MD. Visa encontrar padrões que descrevam os dados e sejam interpretáveis pelo ser humano. Esse tipo de modelo disponibiliza informações a respeito do relacionamento entre os dados, como, por exemplo "peso e idade, em conjunto, são os principais fatores de ocorrência da doença x".

Modelo Preditivo: tipo de modelo gerado pelas técnicas de MD. É criado a partir de um conjunto de variáveis conhecidas e é utilizado para prever valores futuros de outras variáveis de interesse. A partir desse tipo de modelo é possível responder a perguntas como "Essa transação é fraudulenta?" ou "Quanto esse cliente irá produzir de lucro?"

Regra Complexa: utilizadas para inferir candidatos a padrão mais complexos do que o subesquema apresentado no conseqüente da regra. Podem apresentar elementos fracos no antecedente e nem todos os elementos fortes do antecedente estão presentes no conseqüente da regra. Sua interpretação combina informações tanto do antecedente quanto do conseqüente para identificar o candidato a padrão, exigindo a aplicação do conhecimento objetivo do especialista em projeto de BDG, ou seja, domínio dos construtores dos modelos de dados e das regras de combinação dos mesmos.

Regra Simples: apresenta no conseqüente todos os elementos fortes presentes no seu antecedente. Além disso, não existe elemento fraco no seu antecedente. O candidato a padrão encontra-se integralmente descrito no conseqüente da regra.

Regras de Interesse: subconjunto das regras resultantes da MD consideradas úteis.

Regressão: tarefa executada pela MD muito semelhante à classificação. As duas são diferenciadas apenas pelo tipo de dado retornado como saída. No caso da regressão, não são obtidos valores discretos, representando rótulos de classe, mas sim valores contínuos.

Suporte (da regra): corresponde ao percentual de transações apresentadas na entrada que contém todos os itens apresentados tanto no conseqüente quanto no antecedente da regra (A U C).

Anexo 1 Outros tipos de Regras Complexas de Interesse

A seguir são apresentados outros tipos de regras complexas consideradas de interesse, complementarmente àquelas mostradas na seção 5.1.12.

a) Tipo C1: identifica padrões formados por um pacote contendo duas ou mais classes.
C1.ANTEC::= Classe$_{(1)}$, "+", Classe $_{(1)}$-Pacote, {"+" Classe $_{(1)}$-Pacote};
C1.CONSEQ ::= Classe$_{(1)}$-Pacote; S% e CF%.
Onde a classe do item Classe$_{(1)}$ e o pacote do(s) item(ns) Classe$_{(1)}$-Pacote devem ser os mesmos apresentados no conseqüente da regra.
Esse tipo de regra indica que em S% dos esquemas analisados o pacote apresentado contém as classes mostradas na regra.
Exemplo desse tipo de regra e do padrão inferido podem ser vistos nas figuras A1.1 e A1.2, respectivamente.

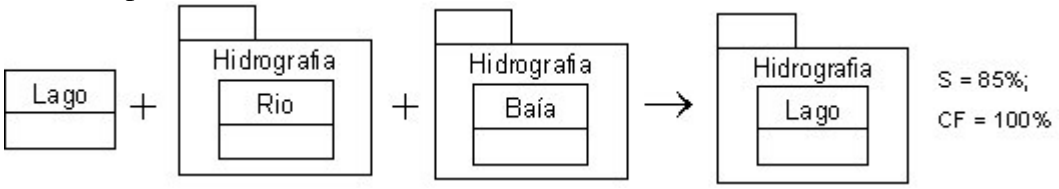


FIGURA A1.1 – Exemplo de regra complexa tipo C1.

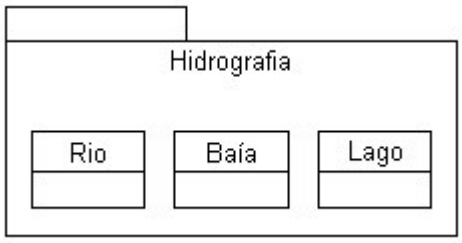


FIGURA A1.2 – Padrão inferido a partir da regra da figura A1.1.

b) Tipo C2: utilizado para inferir padrões formados por uma única instância de classe, qualificada por dois ou mais atributos.
C2.ANTEC::= Atributo$_{(1)}$-Classe, {"+" Atributo $_{(1)}$-Classe};
C2.CONSEQ ::= Atributo$_{(1)}$-Classe; S% e CF%.
Onde as classes do antecedente devem ser, obrigatoriamente, iguais a do conseqüente da regra.
A interpretação desse tipo de regra indica que S% dos esquema analisados apresentam os atributos mostrados na regra, definidos na classe especificada.
Exemplo desse tipo de regra e do padrão inferido podem ser vistos nas figuras A1.3 e A1.4, respectivamente.

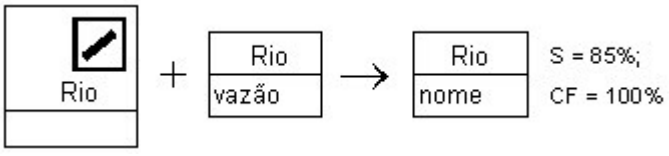


FIGURA A1.3 – Exemplo de regra complexa tipo C2.

FIGURA A1.4 – Padrão inferido a partir da regra da figura A1.3.

c) Tipo C3: identifica padrões formados por uma única instância de classe, contida em pacote e qualificada por dois ou mais atributos.
C3.ANTEC::= Atributo$_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote};
C3.CONSEQ ::= Atributo$_{(1)}$-Classe-Pacote; S% e CF%.
Nesse caso, os pacotes e as classes do antecedente da regra devem ser iguais aos do seu conseqüente.
Esse tipo de regra indica que S% dos esquemas analisados apresentam a classe mostrada na regra definida no pacote especificado, com os atributos apresentados.
Exemplo desse tipo de regra e do padrão inferido podem ser vistos nas figuras A1.5 e A1.6, respectivamente.

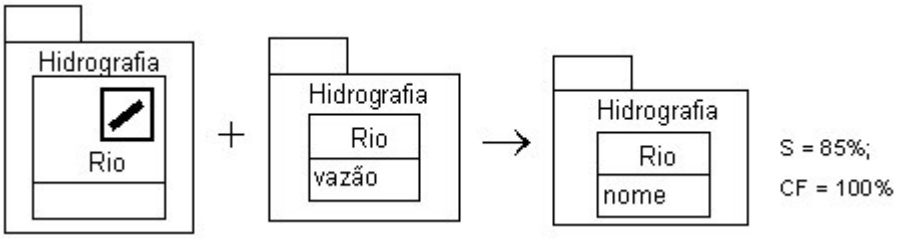


FIGURA A1.5 – Exemplo de regra complexa tipo C3.

FIGURA A1.6 – Padrão inferido a partir da regra da figura A1.5.

d) Tipo C4a: utilizado para identificar padrões formados por duas ou mais classes não relacionadas entre si e definidas em um mesmo pacote. A classe do conseqüente da regra apresenta um atributo.
C4a.ANTEC::= Classe$_{(1)}$-Pacote, "+", Classe$_{(1)}$-Pacote, {"+" Classe $_{(1)}$-Pacote};
C4a.CONSEQ ::= Atributo$_{(1)}$-Classe-Pacote; S% e CF%.
Nesse caso, todos os pacotes presentes na regra devem ser iguais e pelo menos uma das classes do antecedente deve ser igual àquela apresentada no conseqüente da regra.
A partir desse tipo de regra, é possível inferir que S% dos esquemas apresentados na entrada do algoritmo de MD possui o pacote especificado na regra, definido com as várias classes sem relacionamentos entre si. Além disso, a classe do conseqüente possui o atributo apresentado na regra.

Exemplo desse tipo de regra e do padrão por ela identificado são mostrados, na notação UML, nas figuras A1.7 e A1.8.

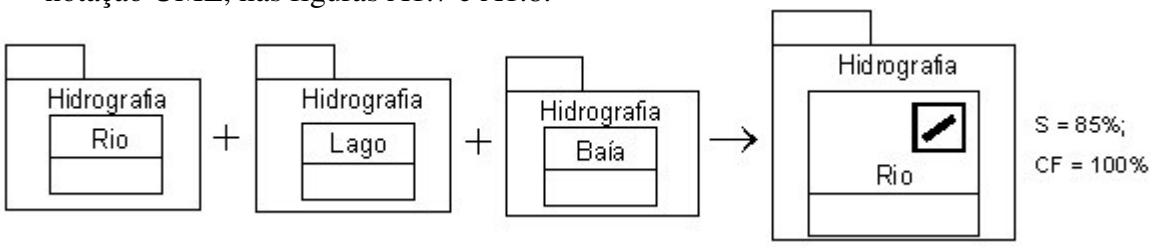


FIGURA A1.7 – Exemplo de regra complexa tipo C4a.

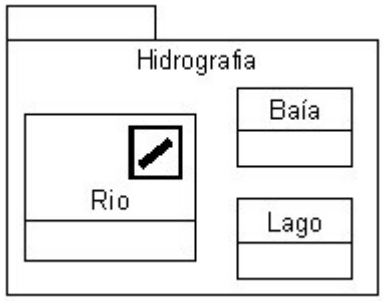


FIGURA A1.8 – Padrão inferido a partir da regra da figura A1.7.

e) Tipo C4b: identifica padrões formados por duas ou mais classes não relacionadas, definidas em um mesmo pacote. A classe do conseqüente da regra apresenta mais de um atributo.
C4b.ANTEC::= Classe$_{(1)}$-Pacote, {"+" Classe $_{(1)}$-Pacote}, "+" Atr ibuto$_{(1)}$-Classe-Pacote, {"+" Atributo $_{(1)}$-Classe-Pacote};
C4b.CONSEQ ::= Atributo$_{(1)}$-Classe-Pacote; S% e CF%.
Nesse caso, todos os pacotes presentes na regra devem ser iguais. As classes dos subesquemas "Atributo $_{(1)}$-Classe-Pacote" devem ser iguais a do conseqüente. As classes dos subesquemas "Classe $_{(1)}$-Pacote" devem ser diferentes da apresentada no conseqüente da regra.
Esse tipo de regra indica que, em S% dos esquemas apresentados na entrada, o pacote especificado na regra contém as várias classes especificadas, sem relacionamentos entre si. Além disso, a classe do conseqüente é qualificada pelos atributos apresentados na regra.
Exemplo desse tipo de regra e do padrão por ela identificado são mostrados nas figuras A1.9 e A1.10.

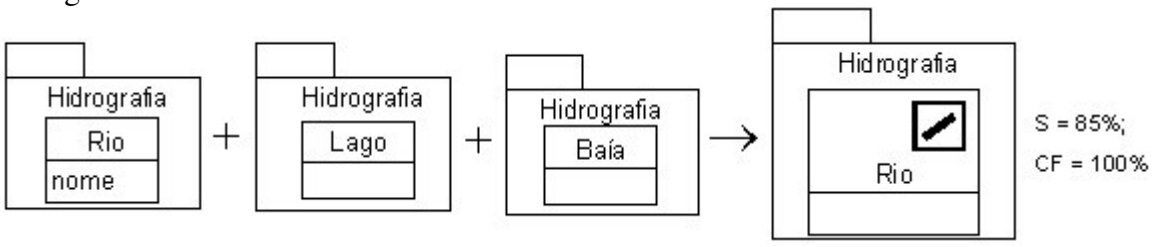


FIGURA A1.9 – Exemplo de regra complexa tipo C4b.

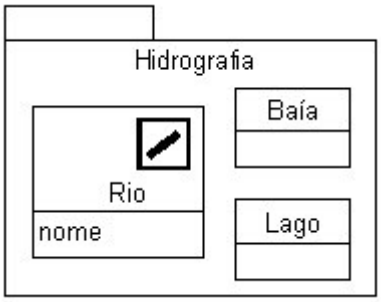


FIGURA A1.10 – Padrão inferido a partir da regra da figura A1.9.

f) Tipo C5b: a partir desse tipo de regra é possível identificar padrões formados por duas classes, definidas em pacotes diferentes e relacionadas entre si através de associação binária.
C5b.ANTEC::= C5a.ANTEC;
C5b.CONSEQ ::= C5a.CONSEQ; S% e CF%.
Nesse caso, os pacotes do antecedente da regra são, obrigatoriamente, diferentes.
Esse tipo de regra indica que em S% dos esquemas analisados as classes do antecedente da regra estão relacionadas entre si através de uma associação binária. Além disso, as duas classes encontram-se definidas nos pacotes conforme apresentados na regra.
Exemplo desse tipo de regra e do padrão por ela identificado são mostrados nas figuras A1.11 e A1.12.

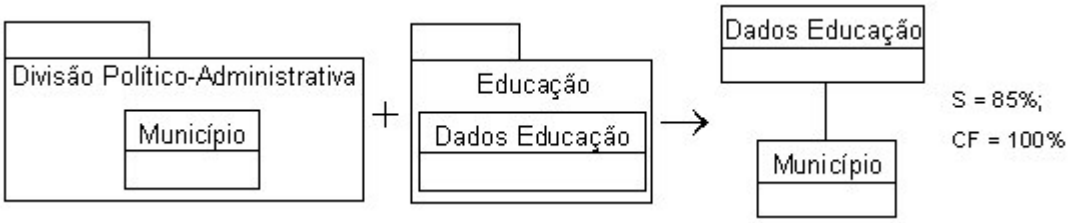


FIGURA A1.11 – Exemplo de regra complexa tipo C5b.

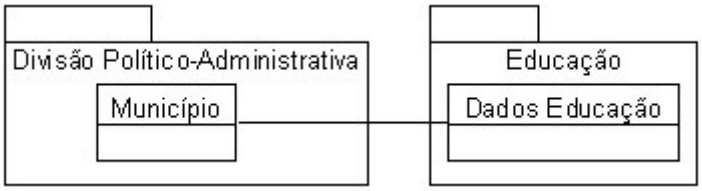


FIGURA A1.12 – Padrão inferido a partir da regra da figura A1.11.

g) Tipo C6a: utilizado para identificar padrões formados por duas classes relacionadas entre si através de uma associação. Uma das classes possui um ou mais atributos.
C6a.ANTEC::= Classe$_{(1)}$, "+", Atributo $_{(1)}$-Classe, {"+" Atributo $_{(1)}$-Classe};
C6a.CONSEQ ::= Associação; S% e CF%.
As classes presentes nos itens do tipo "Atributo $_{(1)}$-Classe" são iguais.
Esse tipo de regra indica que em S% dos esquemas as classes do antecedente estão relacionadas entre si através de uma associação. Além disso, uma das classes é qualificada pelo(s) atributo(s) definido(s) no antecedente.
Exemplos desse tipo de regra e dos padrões por elas identificados são mostrados nas figuras A1.13, A1.14, A1.15 e A1.16.

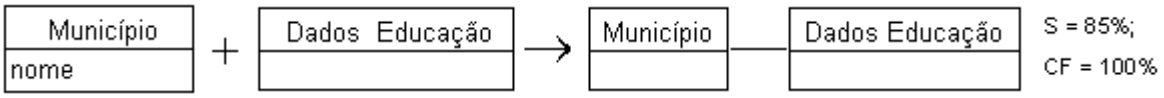


FIGURA A1.13 – Exemplo de regra complexa tipo C6a.

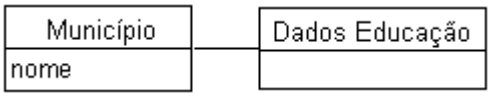


FIGURA A1.14 – Padrão inferido a partir da regra da figura A1.13.

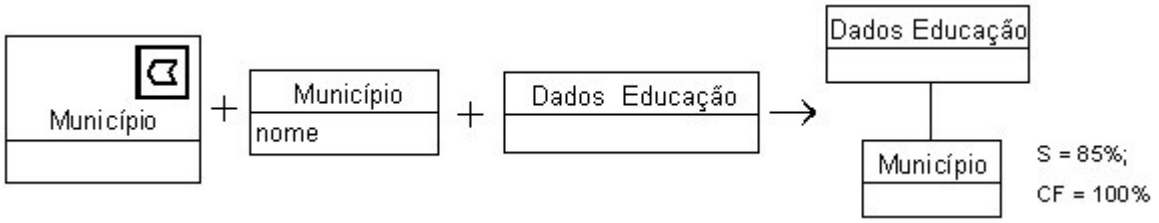


FIGURA A1.15 – Exemplo de regra complexa tipo C6a.

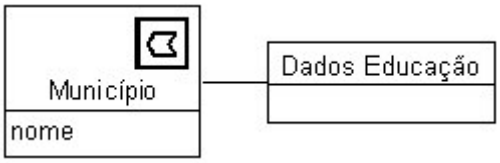


FIGURA A1.16 – Padrão inferido a partir da regra da figura A1.15.

h) Tipo C6b: a partir desse tipo de regra é possível identificar padrões formados por duas classes relacionadas entre si através de uma associação. As duas classes são qualificadas por um ou mais atributos.
C6a.ANTEC::= Atributo$_{(1)}$-Classe, "+", Atributo $_{(1)}$-Classe, {"+" Atributo $_{(1)}$-Classe};
C6a.CONSEQ ::= Associação; S% e CF%.
A leitura desse tipo de regra consiste em que S% dos esquemas apresentados na entrada possui as classes do antecedente relacionadas entre si através de uma associação e são qualificadas pelo(s) atributo(s) definido(s) no antecedente.
Exemplo desse tipo de regra e do padrão por ela identificado são mostrados nas figuras A1.17 e A1.18.

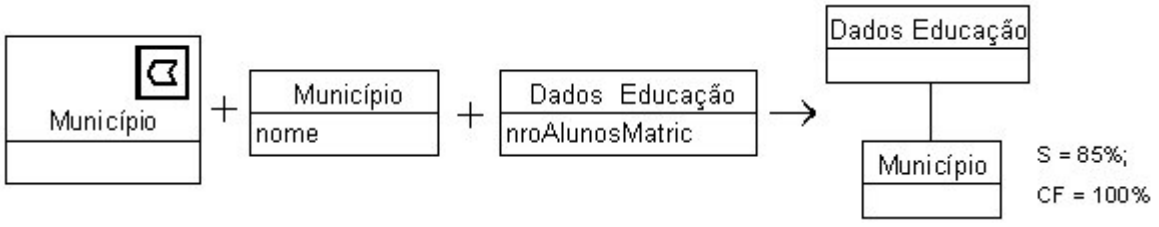


FIGURA A1.17 – Exemplo de regra complexa tipo C6b.

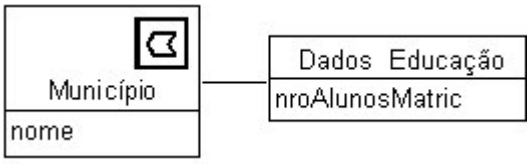


FIGURA A1.18 – Padrão inferido a partir da regra da figura A1.17.

i) Tipo C8: utilizado para identificar padrões formados por três classes e duas associações binárias.
C8.ANTEC::= Classe$_{(1)}$, "+", Associação;
C8.CONSEQ ::= Associação; S% e CF%.
Nesse caso, uma das classes do item Associação presente no antecedente da regra, deve ser igual a uma das classes do conseqüente.
Esse tipo de regra indica que em S% dos esquemas as duas classes do conseqüente estão associadas entre si, além disso uma delas possui uma segunda associação com outra classe.
Exemplo desse tipo de regra e do padrão por ela identificado é mostrado nas figuras A1.19 e A1.20.

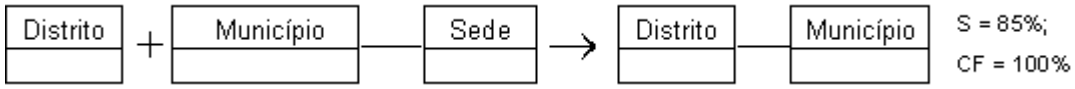


FIGURA A1.19 – Exemplo de regra complexa tipo C8.

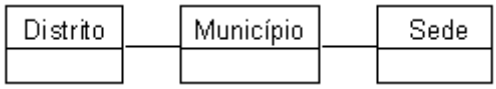


FIGURA A1.20 – Padrão inferido a partir da regra da figura A1.19.

j) Tipo C9: identifica padrões formados por várias classes e associações binárias.
C9.ANTEC::= Classe$_{(1)}$, "+", Associação, {"+" Associação}| Associação, "+", Associação, {'+' Associação};
C9.CONSEQ ::= Associação; S% e CF%.
A leitura desse tipo de regra indica que em S% dos esquemas as classes presentes na regra estão associadas entre si, conforme descrito.
Exemplo de regra desse tipo e do padrão por ela identificado são vistos nas figuras A1.21, A1.22, A1.23 e A1.24.

FIGURA A1.21 – Exemplo de regra complexa tipo C9.

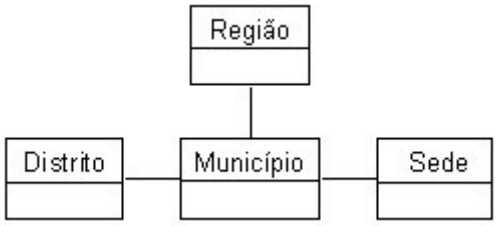


FIGURA A1.22 – Padrão inferido a partir da regra da figura A1.21.

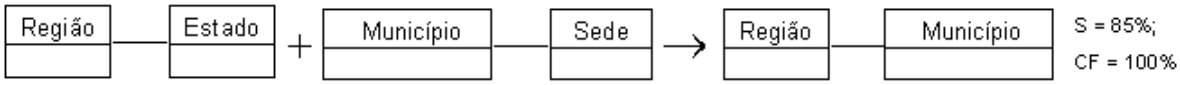


FIGURA A1.23 – Exemplo de regra complexa tipo C9.

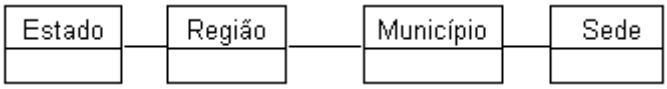


FIGURA A1.24 – Padrão inferido a partir da regra da figura A1.23.

k) Tipo C10a: identifica padrões formados por várias classes associadas entre si, onde pelo menos uma delas é definida dentro de pacote. Sua estrutura é:
C10a.ANTEC::= Classe$_{(1)}$-Pacote, "+", Associação, {"+" Associação}, {"+" Classe$_{(1)}$-Pacote}| Classe$_{(1)}$, "+", Classe $_{(1)}$-Pacote, "+", Associação, {'+' Associação}, {"+"Classe $_{(1)}$-Pacote};
C10a.CONSEQ ::= Associação; S% e CF%.
Nesse caso, o item do tipo Classe$_{(1)}$ só deve estar presente na regra se uma das classes do conseqüente não possui outro relacionamento e não está definida dentro de pacote.
Esse tipo de regra indica que em S% dos esquemas as classes presentes na regra possuem associações entre si e estão definidas em pacotes conforme descrito na regra.
Exemplo de regra desse tipo e do padrão por ela identificado são ilustrados nas figuras A1.25, A1.26, A1.27 e A1.28.

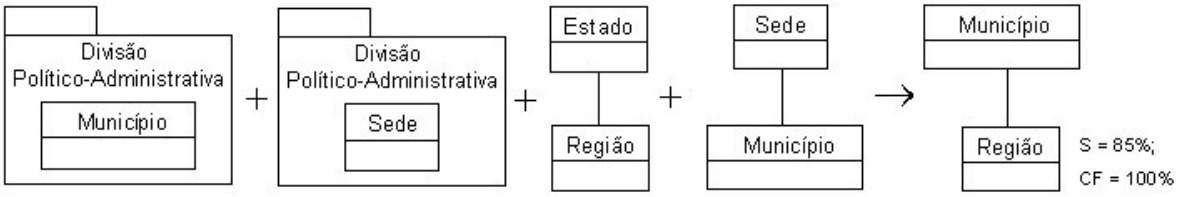


FIGURA A1.25 – Exemplo de regra complexa tipo C10a.

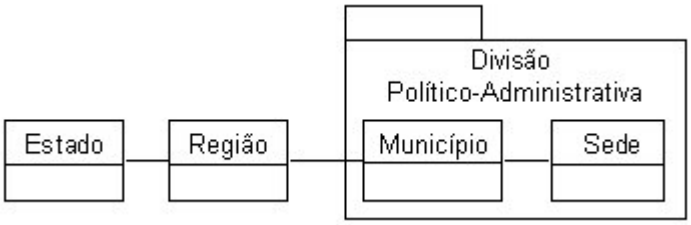


FIGURA A1.26 – Padrão inferido a partir da regra da figura A1.25.

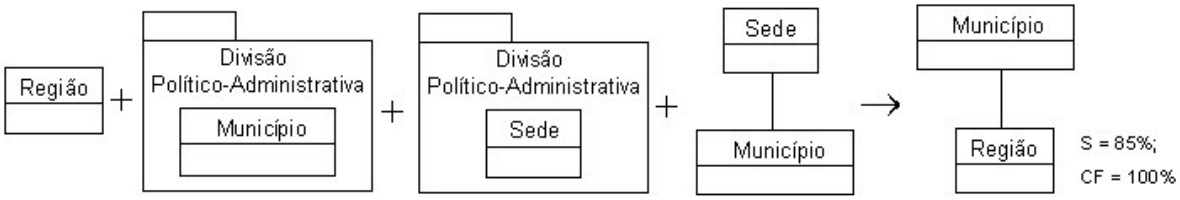


FIGURA A1.27 – Exemplo de regra complexa tipo C10a.

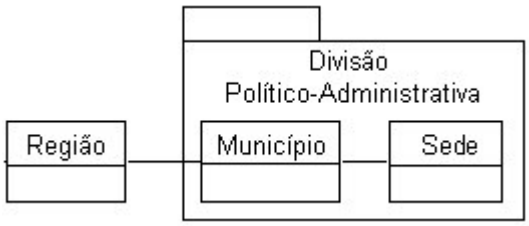


FIGURA A1.28 – Padrão inferido a partir da regra da figura A1.27.

l) Tipo C10b: identifica padrões formados por várias classes associadas entre si, onde pelo menos uma delas é qualificada por, no mínimo, um atributo.

C10b.ANTEC::= Atributo$_{(1)}$-Classe, "+", Associação, {"+", Associação}, {"+" Atributo$_{(1)}$-Classe}| Classe$_{(1)}$, "+", Atributo $_{(1)}$-Classe, "+", Associação, {"+" Associação}, {"+"Atributo $_{(1)}$-Classe};
C10b.CONSEQ ::= Associação; S% e CF%.
O item do tipo Classe$_{(1)}$ só deve estar presente na regra se uma das classes do conseqüente não possui outro relacionamento e não é qualificada por atributo(s).
Esse tipo de regra indica que em S% dos esquemas as classes presentes na regra possuem associações entre si e são qualificadas por atributos conforme descrito na regra.
Exemplos desse tipo de regra e dos padrões por elas identificados são mostrados nas figuras A1.29, A1.30, A1.31 e A1.32.

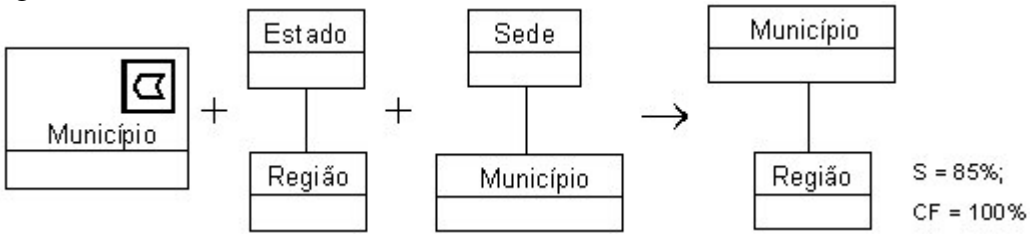


FIGURA A1.29 – Exemplo de regra complexa tipo C10b.

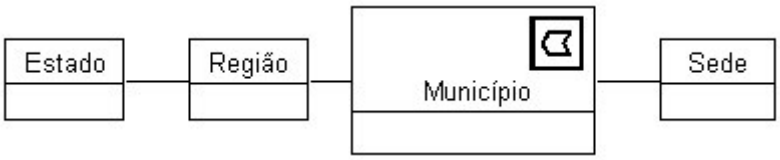


FIGURA A1.30 – Padrão inferido a partir da regra da figura A1.29.

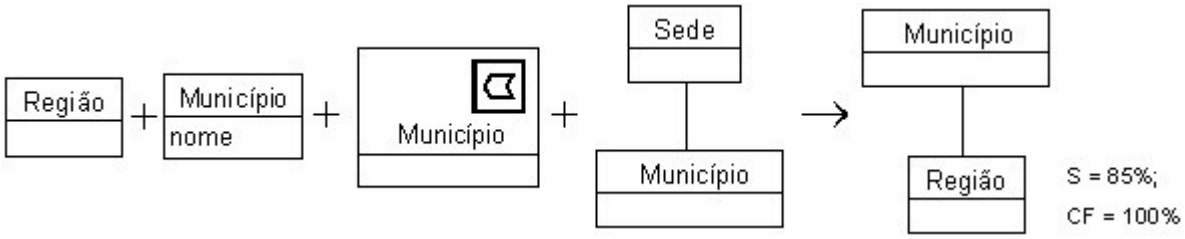


FIGURA A1.31 – Exemplo de regra complexa tipo C10b.

FIGURA A1.32 – Padrão inferido a partir da regra da figura A1.31.

Anexo 2 Programa de Geração de Esquemas

O algoritmo foi desenvolvido na linguagem pascal, pois já se tinha domínio da mesmo, o que facilitou e agilizou o desenvolvimento.

O objetivo do algoritmo consiste em gerar dados de entrada para uma ferramenta de mineração de dados que busque identificar associações entre itens freqüentes. Tais dados, no contexto desse trabalho, consistem em esquemas de bancos de dados.

1. Dados de entrada do algoritmo

Para verificar se a ferramenta de mineração de dados realmente indica o número de ocorrências correto para os itens freqüentes dentro do arquivo de entrada da ferramenta (que corresponde ao arquivo de saída do algoritmo), alguns conjuntos de itens, juntamente com sua freqüência, devem ser informados pelo usuário. Assim, são passados como parâmetros de entrada:

- Total de esquemas: corresponde ao número total de esquemas a serem gerados.
- Relacionamentos freqüentes: especificação de, no máximo, dez relacionamentos que devem aparecer com uma freqüência mínima, especificada pelo usuário.
- Número de classes/esquema: quantidade máxima de classes que cada esquema possui.
- Número máximo de atributos/classe: quantidade máxima de atributos que cada classe possui.
- Nome do arquivo de saída: nome do arquivo que conterá os dados gerados.

A informação sobre os relacionamentos freqüentes é composta pela definição do relacionamento e a freqüência com que ele ocorre no arquivo de saída. O relacionamento deve possuir, no máximo, cinco classes e é indicado em uma notação própria, qual seja:

<nome_classe1>#<nome_classe2>#<nome_classe3>>#<nome_classe4>#
<nome_classe5>#assoc

Como só estão sendo tratados relacionamentos de associação, a ordem de apresentação das classes não modifica a semântica, por este motivo, os relacionamentos gerados na saída serão apresentados com as classes em ordem alfabética.

Apesar de aceitar, como parâmetro de entrada, relacionamentos entre três classes, o algoritmo gera, no arquivo de saída, sempre relacionamentos dois a dois, isto é, na saída só serão apresentados relacionamentos entre duas classes. A lógica de construção dos relacionamentos consiste em associar a primeira classe com a segunda e esta com a terceira, por exemplo:

Entrada: b# a #c ###assoc
Saída: a#b#assoc e a#c#assoc.

A freqüência dos relacionamentos deve ser informada em termos de percentual. Caso o relacionamento que está sendo especificado contenha outro, deve ser informado também o índice do vetor correspondente ao relacionamento contido nele. Se ele estiver contido em outro, o relacionamento que o contém deve ser especificado. Esta característica restringe que cada relacionamento contenha apenas um outro ou esteja contido em apenas um relacionamento.

Se for necessário especificar, no parâmetro de entrada, um relacionamento entre um número menor que cinco classes, o nome das demais classes é omitido, da seguinte forma:

b#a####assoc

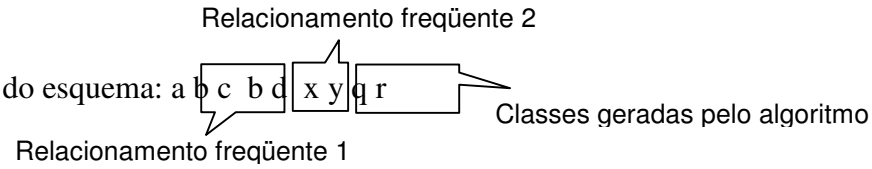
2. Geração dos dados

Para cada esquema, primeiro são incluídas as classes que fazem parte de relacionamentos freqüentes. O algoritmo ainda gera, randomicamente, novas classes, desde que o número total não exceda ao máximo de classes/esquema especificado pelo usuário.

Para adicionar as classes que fazem parte de relacionamentos freqüentes ao esquema, o algoritmo verifica, na ordem em que foram informados, todos os relacionamentos freqüentes e adiciona cada classe ao esquema até atingir o número máximo de classes/esquema ou até que todas as classes dos relacionamentos freqüentes já tenham sido colocadas no esquema. Com isso, os primeiros esquemas gerados possuirão, em geral, mais de um relacionamento freqüente, dentre os especificados pelo usuário.

Após identificar todas as classes do esquema, o algoritmo gera, randomicamente, para cada classe, seus atributos. O número de atributos por classe é determinado randomicamente pelo algoritmo, obedecendo apenas o número máximo de atributos/classe especificado pelo usuário.

De posse de todas as classes com seus atributos, são gerados os relacionamentos entre as classes do esquema. O algoritmo insere os relacionamentos freqüentes correspondentes às classes que os compõem e associa as classes geradas randomicamente, pegando sempre a do índice atual com a do índice seguinte no vetor de classes do esquema. Por exemplo:



Relacionamentos gerados:

a#b#assoc; b#c#assoc; b#d#assoc
x#y#assoc; y#q#assoc; q#r#assoc

3. Arquivo de Saída

Os dados de saída são gerados em duas colunas, a primeira corresponde ao número do esquema e a segunda contém os itens da transação. Um único esquema gera várias transações.

O arquivo de saída é organizado no formato de entrada da ferramenta Intelligent Miner for Data. Assim, uma transação ocupa várias linhas do arquivo e o nome das colunas não é especificado. Como requisito da ferramenta, o tamanho das colunas é fixo e, por isso, o algoritmo preenche os espaços deixados pelos itens com brancos. As colunas são separadas por espaços em branco sem vírgulas e a primeira coluna (com o identificador do esquema) é usada como identificador da transação.

4. Algoritmo com parâmetros de entrada já especificados

```
program Gera_Dados;

uses crt, dos;

const

      {tamanho do vetor 'relacs-freq'}

      tam_relacs_freq=3;

type

    {registro que especifica informacoes necessarias a respeito de
relacionamentos frequentes especificados pelo usuario}

    relacionamentos = record

        relac:                                                  string;
{<nome_classe1>#<nome_classe2>#<nome_classe3>#<nome_tp_relac>.

                              Vai aceitar 3, mas fara sempre o 1$ com
o 2$ e o 2$ com o 3$}

        freq: real;             {freq. em que aparece no conjunto de
entrada, em %}

        no_pend: integer;     {no. de vezes que ainda falta ser gerado
para atingir o percentual acima}

        i_subconj: integer;  {Indice do subconjunto, especificado pelo
user. Deve ser zero, caso nao possua subconjunto.

                              Considerar      subconjunto      apenas
associacoes que impliquem em repeticoes. Por exemplo, `ab` e `bc`

                              sao subconjunto de `abc`, mas `ac` nao}

        i_superconj: integer;{indice do superconjunto, especificado
pelo user. Deve ser zero, caso nao possua superconjunto}

    end;


    {registro que especifica informacoes necessarias a respeito de
classes frequentes.

    A informacao e obtida a partir dos relacionamentos frequentes}

    classes = record

        classe: string;     {nome da classe frequente}

        no_pend: integer;    {qtde de vezes que ainda falta ser gerada
para atingir o percentual dos relacs onde ela aparece}

        subconj:  boolean;       {indica  se  ela  pertence  a  algum
subconjunto ou nao}

    end;
```

```
    {registro que simula uma tabela com o relacionamento nxn entre
relacionamentos e classes}
    class_relac=record
        nome_classe: string;     {nome da classe}
        ind_RelacFreq:  integer;     {indice  no  vetor  de  classes
frequentes}
    end;


    {registro que guarda o nome da classe e de um de seus atributos}
    classe_atrib = record
        classe: string;          {nome da classe}
        atrib: string;           {nome do atrib}
    end;


var tot_esq: integer;                                          {numero
total de esquemas}
    relacs_freq:        array[1..10]        of        relacionamentos;
{especificacao das construcoes que devem ser frequentes}
    max_atribs: integer;                                       {numero
maximo de atributos por classe}
    max_classes: integer;                                      {numero
maximo de classes por esquema}
    w,z,x,v,j: integer;                                      {contador
do laco}
    classes_freq: array[1..30] of classes;                   {guarda as
classes freq especificadas nos relacionamentos}
    tamMaxClasses_freq: integer;                               {tamanho
maximo do vetor com as classes frequentes}
    tamClassesFreq:integer;                                    {tamanho
atual do vetor de classes frequentes}
    classes_esq: array[1..10] of string;                     {vetor com
os nomes das classes que compoem o esquema}
    classes_relacionamentos: array[1..30] of class_relac;   {funciona
como uma tabela de relacionamento com info sobre as
                                                            classes   e
a quais relacs elas pertencem}
    tamClasses_relacionamentos: integer;                     {tamanho do
vetor que guarda a relacao entre as classes e
                                                            os
relacionamentos onde elas estao}
    qtdeClassesEsq: integer;                                   {qtde de
classes existente atualmente no vetor de classes do esq}
    naopode:integer;                                         {indice do
relacionamento que nao pode ser usado novamente para
```

```
                                                                buscar
classes frequentes}
    h,m,s,ss: word;                                              {Usadas
para armazenar a hroa do sistema}


    classes_atribs: array[1..40] of classe_atrib;             {vetor que
contem as classes e todos os seus atributos}
    tamClasses_atribs: integer;                                 {tamanho
atual do vetor de classes e atributos}
    relacs_classe: array[1..10] of integer;                   {guarda os
indices dos relacionamentos de uma classe}
    tamRelacs_classe: integer;                                  {tamanho
atual do vetor com os relacs da classe}
    qtdeRelacFreqPend: integer;
    teste: boolean;
    relacs_usados: array[1..10] of integer;                  {indice dos
relacionamentos em `relacs_freq` usados pela classe}
    tamRelacs_usados: integer;
    relacs_esq: array[1..15] of string;                          {guarda
todos os elementos do esquema atual}
    tamRelacs_esq: integer;                                  {tamanho do
vetor `relacs_esq`}
    arqSai: text;                                            {arquivo de
saida}
    nome_arqSai: string;                                        {nome do
arquivo de saida}
    reg: string;                                              {linha que
sera escrita no arquivo de saida}
    no_esq: string;
    item: string[15];                                           {item do
arquivo de saida}
    tam: integer;


{Coloca todas as classes de relacionamentos frequentes e o id do
relacionamento no vetor com as associacoes
 entre classes e relacionamentos}
{Usado por }
procedure atualizaClasses_relacionamentos;
var i: integer;
    classe, relac: string;
begin
    for i:=1 to tam_Relacs_Freq do
       begin
          relac:= relacs_freq[i].relac;
          for j:=1 to 3 do
```

```
            begin
              classe:= copy(relac,1,pos('#', relac)-1);
              {remove a classe 'h' do relacionamento}
              delete(relac, 1, pos('#', relac));
              if classe<>''
              then begin

tamClasses_relacionamentos:=tamclasses_relacionamentos+1;

classes_relacionamentos[tamClasses_relacionamentos].nome_classe:=
classe;

classes_relacionamentos[tamClasses_relacionamentos].ind_RelacFreq:= i;
                  end;
            end;
       end;
end;


{procura a classe 'classe' no vetor 'classes_relacionamentos'.
 Retorna o indice da classe `classe` no vetor ou `zero` se ela nao
estiver no vetor}
{Usado por }
function pesquisaClasses_relacs (classe: string): integer;
var i: integer;
begin
   pesquisaClasses_Relacs:=0;
   for i:=1 to tamClasses_relacionamentos do
     begin
        if classes_relacionamentos[i].nome_classe = classe
        then                pesquisaClasses_relacs                  :=
classes_relacionamentos[i].ind_relacFreq;
     end;
end;


{Procura a classe 'classe' no vetor de classes frequentes e retorna
seu indice ou zero, caso ela nao esteja no vetor}
function achaClasseFreq(classe: string): integer;
var i: integer;
begin
   achaClasseFreq:=0;
   for i:=1 to tamClassesFreq do
       begin
          if classes_freq[i].classe = classe
```

```
          then begin
                 achaClasseFreq:= i;
                 exit;
               end;
       end;
end;

{verifica se existe algum relacionamento frequente com no_pendencias
<> 0.
 Retorna o indice, no vetor `relacs_freq`, do ultimo relacionamento
com no_pendencias<>0 ou `zero`,
 se todos tiverem no_pendencias=0}
{Usado por `verClassesFreq`}
function verQtdeRelacFreqPend: integer;
var i: integer;
    cont: integer;
begin
   verQtdeRelacFreqPend:= 0;
   cont:=0;
   for i:=1 to tam_relacs_Freq do
     begin
        if relacs_freq[i].no_pend>0
        then cont:= cont+1;
     end;
     verQtdeRelacFreqPend:= cont;
end;

{coloca zero em todos os indices de `relacs_usados` e em
`tamRelacs_usados`}
procedure zeraRelacs_usados;
var i: integer;
begin
   for i:=1 to 10 do
      relacs_usados[i]:=0;

   tamRelacs_usados:= 0;
end;

{verifica se o relacionamento `relac` ja foi usado no esquema atual}
function verRelacs_usados(relac:integer):boolean;
var i: integer;
```

```
begin
   verRelacs_usados:= false;
   for i:=1 to tamRelacs_usados do
      begin
         if relacs_usados[i]= relac
         then begin
                 verRelacs_usados:= true;
                 exit;
              end;
      end;
end;

{verifica a qtde de classes que o relacionamento de indice `i_relac`
em `relacs_freq` possui}
function qtdeClassesRelacFreq(i_relac: integer):integer;
var i, cont: integer;
begin
    cont:= 0;
    for i:=1 to tamClasses_relacionamentos do
       begin
          if classes_relacionamentos[i].ind_relacFreq = i_relac
          then cont:= cont+1;
       end;

    qtdeClassesRelacFreq:= cont;
end;

{coloca as classes do relacionamentos no indice `ind` em `relacs_freq`
em `classes_esq`}
procedure colocaRelacEmClasses_freq(ind: integer);
var j, iCLfreq, iSub: integer;
    dec: boolean;
begin
    relacs_freq[ind].no_pend:=relacs_freq[ind].no_pend - 1;
    if relacs_freq[ind].i_subconj<>0
    then begin
            iSub:= relacs_freq[ind].i_subconj;
            relacs_freq[iSub].no_pend:= relacs_freq[iSub].no_pend - 1;
         end;

    for j:=1 to tamClasses_relacionamentos do
```

```
       if classes_relacionamentos[j].ind_relacFreq = ind
       then begin
              qtdeClassesEsq:=qtdeClassesEsq+1;
              {coloca a classe no vetor de classes do esquema}
              classes_esq[qtdeClassesEsq]:=
classes_relacionamentos[j].nome_classe;
              {atualiza pendencias das classes frequentes}
              iCLfreq:=
achaClasseFreq(classes_relacionamentos[j].nome_classe);
              classes_freq[iCLfreq].no_pend:=
classes_freq[iCLfreq].no_pend - 1;
           end;
end;

{coloca algumas classes frequentes no vetor de classes do esquema.
 Pega na ordem do vetor `relacs_freq` ate a qtde que nao exceda a
maxima do esquema}
procedure usaClassesFreq;
var i, totalCL: integer;
begin
   zeraRelacs_usados;
   for i:=1 to tam_relacs_freq do
      begin
         if relacs_freq[i].no_pend<> 0
         then if relacs_freq[i].i_superconj=0
              then begin
                      if verRelacs_usados(i) = false
                      then begin
                             totalCL:=    qtdeClassesRelacFreq(i)    +
qtdeClassesEsq;
                             if totalCL < max_Classes
                             then begin
                                     colocaRelacEmClasses_freq(i);

tamRelacs_usados:=tamRelacs_usados+1;

Relacs_usados[tamRelacs_usados]:=i;
                                  end;
                           end
                      end
              else  if  verRelacs_usados(relacs_freq[i].i_superconj)  =
false
                   then begin
```

```
                        totalCL:=      qtdeClassesRelacFreq(i)      +
qtdeClassesEsq;
                        if totalCL < max_Classes
                        then begin
                               colocaRelacEmClasses_freq(i);
                               tamRelacs_usados:=tamRelacs_usados+1;
                               Relacs_usados[tamRelacs_usados]:=i;
                             end;
                      end;
      end;
end;

{verifica se a classe 'classe' encontra-se no vetor 'classes_freq' de
tamanho 'tam_vetor'.
 Retorna o indice da classe no vetor ou `zero`, se ela nao estiver
presente}
{Usada por }
function pesquisaClasse(classe: string; tam_vetor: integer): integer;
var i: integer;
begin
    pesquisaClasse:= 0;
    for i:=1 to tam_vetor do
       begin
          if classes_freq[i].classe=classe
          then pesquisaClasse:=i;
       end;
end;

{inicializa a var no_pend de todos os registros das construcoes
frequentes de acordo com o percentual especificado}
procedure inicializaNro_pendRelacs;
var i:integer;
begin
   for i:=1 to tam_relacs_freq do
      begin
         relacs_freq[i].no_pend:=           round((tot_esq           *
relacs_freq[i].freq)/100);
      end;
end;

{pode ser inserido em atualizaClasses_relacionamentos}
{le os relacionamentos frequentes e atualiza o vetor `classes_freq`}
```

```
procedure inicializaClasses_freq;
var j: integer;  {posicao no vetor de relacionamentos frequentes}
    i: integer; {posicao no vetor de classes frequentes}
    relac_temp:  string;  {relacionamento  da  posicao  'j'  do  vetor
relacs_freq}
    nom_classe: string; {nome da classe lida no contador 'i' atual}
    h: integer;
    indClasses_relac:  integer;  {se  a  classe  já  estiver  no  vetor,
guarda o indice dela}

begin
   qtdeClassesEsq:= 0;

   {para a qtde de relacionamentos frequentes}
   for j:=1 to tam_relacs_freq do
      begin
         if j=1
         then i:=0;

         relac_temp:= relacs_freq[j].relac;
         {3  e  a  qtde  maxima  de  classes  relacionadas  que  sera
especificada}
         for h:=1 to 3 do
            begin
              {pega o nome da classe 'h'}

              nom_classe:= copy(relac_temp,1,pos('#', relac_temp)-1);
              {remove a classe 'h' do relacionamento}
              delete(relac_temp, 1, pos('#', relac_temp));

              if nom_classe<>''
              then begin
                     if  i<>  0  {vetor  de  classes  frequentes  nao  e
vazio}
                     then {se  a  classe  ainda  nao  esta  no  vetor  de
classes frequentes,
                           a  coloca  com  a  qtde  de  vezes  que  deve
aparecer}
                          begin
                             indClasses_relac:=
pesquisaClasse(nom_classe, i);
                             if indClasses_relac = 0
                             then begin
```

```
tamClassesFreq:=tamClassesFreq+1;
                                        i:= i + 1;
                                        classes_freq[i].classe:=
nom_classe;
                                    end
                            end
                        else  {nao  existe  classe  no  vetor  de  classes
freq}
                            begin
                               tamClassesFreq:=tamClassesFreq+1;
                               i:=1;
                               classes_freq[i].classe:= nom_classe;
                            end;
                    end;
            end;
      end;
end;


{verifica se a classe 'classe' existe no vetor de classes do esquema.
`Tam` e a qtde de classes que existe no vetor}
{Usada por }
function verClasses_esq(classe: string; tam: integer): boolean;
var i:integer;
begin
   verClasses_esq:= false;
   for i:=1 to tam do
     begin
       if classes_esq[i]=classe
       then verClasses_esq:=true;
     end;
end;


{Gera aleatoriamente o restante das classes que faltam para completar
o maximo de classes do esquema.
 Comeca do indice onde as classes frequentes terminam `qtdeClassesEsq`
}
procedure geraClasses;
var i:integer;
    classe: integer;   {'nome` da classe gerada}
    classeStr: string; {nome da classe gerada}
```

```
begin
    i:=0;
    for i:= (qtdeClassesEsq+1) to max_classes do          {gera a qtde
de classes especificada em max}
       begin
         repeat
           randomize;
           classe:= random(200);
           str(classe, classestr);
         until (classe<>0) and (verClasses_esq(classestr, i)=false);

         classes_esq[i]:= classeStr;
         qtdeClassesEsq:= qtdeClassesEsq + 1;
       end;
end;

{verifica se `classe` pertence ao relacionamento (que e um subconj) de
indice `iSub` em `relacs_freq`
 Retorna true, se sim}
function verClasseSubconj (classe: string; iSub: integer): boolean;
var i: integer;
begin
   verClasseSubconj:= false;
   for i:=1 to tamClasses_relacionamentos do
      begin
         if     (classes_relacionamentos[i].nome_classe=classe)     and
(classes_relacionamentos[i].ind_relacFreq=iSub)
         then begin
                verClasseSubconj:= true;
                exit;
              end;
      end;
end;

{coloca zero em todas as posicoes do vetor `relacs_classe` e em
`tamRelacs_Classe`}
procedure zeraRelacs_classe;
var i: integer;
begin
   tamRelacs_classe:=0;
   for i:=1 to 10 do
      relacs_classe[i]:=0;
```

```
end;


{coloca na variável global 'relacs_classes' a relação de todos os
indices em 'relacs_freq' onde a classe
'classe' aparece. Retorna a qtde de relacionamentos onde a classe
aparece}
function verRelacs_Classe (classe: string): integer;
var cont, i: integer;


begin
   cont:= 0;


   zeraRelacs_classe;
   for i:= 1 to tamClasses_relacionamentos do
      begin
      if classes_relacionamentos[i].nome_classe = classe
      then begin
            cont:=cont+1;

      relacs_classe[cont]:=classes_relacionamentos[i].ind_relacFreq;
           end;
      end;


   tamRelacs_classe:= cont;
   verRelacs_Classe:= cont;
end;



procedure atualizaNo_pendClasses;
var x, i, j, iRel: integer;


begin
   for x:=1 to tamClassesFreq do
     begin
      if verRelacs_classe (classes_freq[x].classe)>1
      then begin
            {pega o no_pend do 1° relac onde a classe aparece, caso ele
seja um subconjunto}
                iRel:= relacs_classe[1];
            if relacs_freq[iRel].i_subconj = 0
            then classes_freq[x].no_pend:= relacs_freq[iRel].no_pend;
```

```
            for i:= 2 to tamRelacs_classe do
              begin
                 irel:= relacs_classe[i];

                 if relacs_freq[irel].i_subconj = 0
                 then {e um subconjunto. Pega no_pend do sub}
                          classes_freq[x].no_pend:=
classes_freq[x].no_pend + relacs_freq[irel].no_pend
                      else                                              if
verClasseSubconj(classes_freq[x].classe,relacs_freq[irel].i_subconj)  =
false
                          then  {e  superconjunto,  mas  a  classe  em
questao nao pertence ao subconjunto dele}

classes_freq[x].no_pend:=classes_freq[x].no_pend+
relacs_freq[irel].no_pend;


              end;
           end
      else begin
                 iRel:= relacs_classe[1];
                 classes_freq[x].no_pend:=    relacs_freq[iRel].no_pend;
{só aparece uma vez}
             end;
     end;
end;


{Verifica  se  o  atributo  `atrib`  existe  no  vetor  `classes_atribs`.
'Ini' e o indice onde comecam os atributos da classe atual no
 vetor  de  classes  e  atributos.  'Tam'  e  o  tamanho  do  vetor
'classes_atribs' que deve ser passado pq so e atualizado apos a
 chamada da funcao. }
{Usada por `GeraAtributos`}
function  verAtributo(atrib:  string;  ini:  integer;  tam:integer):
boolean;
var i: integer;


begin
   verAtributo:= false;
   for i:=ini to tam do
     begin
       if classes_atribs[i].atrib = atrib
       then begin
              verAtributo:=true;
```

```
              exit;
            end;
     end;
end;


{gera atributos para a classe `classe`. `Ini` e o ultimo indice usado
no vetor `classes_atribs`
 Retorna o tamanho final do vetor `classes_atribs` apos a insercao dos
atributos da classe `classe`
 Se a qtde de atributos para a classe for `zero`, ela nem aparece no
vetor `classes_atribs`}
function geraAtributos (classe: string; ini: integer):integer;
var i: integer;
    qtde_max: integer;   {qtde maxima de atributos para essa classe.
Obtida aleatoriamente}
    tam: integer;       {tamanho do vetor de classes e atributos}
    nome_atrib: string;
    atributo: string;
begin
   max_atribs:= 5;
   randomize;
   if odd(random(500))
   then qtde_max:= random(max_atribs)
   else qtde_max:= random(max_atribs)+1;
   tam:= ini;

   for i:=1 to qtde_max do
      begin
         repeat
           randomize;
           str(random(100), nome_atrib);
           atributo:=classe+': '+nome_atrib;
         until verAtributo(atributo, ini, tam) = false;

         tam:=tam+1;
         classes_atribs[tam].classe:= classe;
         classes_atribs[tam].atrib:=atributo;
      end;

   geraAtributos:=tam;
end;
```

```
{apaga todas as posicoes do vetor de relacionamentos do esquema e
coloca zero em tamRelacs_esq}
procedure zeraRelacs_esq;
var i: integer;
begin
   tamRelacs_esq:= 0;

   for i:=1 to 15 do
      relacs_esq[i]:= '';
end;

procedure geraRelacionamentos;
var x, j, h, i, classeH, classeH1: integer;

begin
   zeraRelacs_esq;
   x:= 1;
   for j:=1 to tamRelacs_usados do
      begin
         i:= QtdeClassesRelacFreq(j);
         for h:= 1 to (i-1) do
            begin
               tamRelacs_esq:= tamRelacs_esq + 1;
               if (classes_esq[x] < classes_esq[x+1])
               then relacs_esq[tamRelacs_esq]:= classes_esq[x] + '#' +
classes_esq[x+1] + '#assoc'
               else relacs_esq[tamRelacs_esq]:= classes_esq[x+1] + '#'
+ classes_esq[x] + '#assoc';
               x:= x + 1;
            end;
         x:= x+1;
      end;

   for h:=x to (qtdeClassesEsq - 1) do
      begin
        tamRelacs_esq:= tamRelacs_esq + 1;
        {tive que transformar em numeros, pois na comparacao de string
'82'>'138'}
        val(classes_esq[h], classeH, classeH);
        val(classes_esq[h+1], classeH1, classeH1);
        if (classeH < classeH1)
```

```
        then  relacs_esq[tamRelacs_esq]:=  classes_esq[h]  +  '#'  +
classes_esq[h+1] + '#assoc'
        else  relacs_esq[tamRelacs_esq]:=  classes_esq[h+1]  +  '#'  +
classes_esq[h] + '#assoc';
      end;
end;

{verifica se o atributo 'atrib' existe no vetor de atributos. `Tam` e
a qtde de atribs que existe no vetor}
{Usada por }
function verAtribs_esq(atrib: string; tam: integer): boolean;
var i:integer;
begin
   verAtribs_esq:= false;
   for i:=1 to tam do
     begin
       if classes_atribs[i].atrib=atrib
       then verAtribs_esq:=true;
     end;
end;

{programa principal}
begin
    for w:=1 to 40 do
      begin
         classes_esq[w]:='';
         classes_freq[w].classe:='';
         classes_freq[w].no_pend:=0;
         classes_freq[w].subconj:=false;
         classes_relacionamentos[w].nome_classe:='';
         classes_relacionamentos[w].ind_relacFreq:=0;
         relacs_freq[w].relac:='';
         relacs_freq[w].freq:=0;
         relacs_freq[w].no_pend:=0;
         relacs_freq[w].i_subconj:=0;
         classes_atribs[w].classe:='';
         classes_atribs[w].atrib:='';
      end;

    {parametros de entrada}
    tot_esq:= 10;
```

```
relacs_freq[1].relac:= 'c#b#a#assoc';
relacs_freq[1].freq:= 40;
relacs_freq[1].i_subconj:= 3;
relacs_freq[1].i_superconj:= 0;

relacs_freq[2].relac:= 'd#b#assoc';
relacs_freq[2].freq:= 70;
relacs_freq[2].i_subconj:= 0;
relacs_freq[2].i_superconj:= 0;

relacs_freq[3].relac:= 'a#b##assoc';
relacs_freq[3].freq:= 60;
relacs_freq[3].i_subconj:= 0;
relacs_freq[3].i_superconj:= 1;

max_atribs:= 6;
max_classes:= 8;

nome_arqSai:='teste.txt';

{fim dos parametros de entrada}

tamClasses_relacionamentos:=0;
tamClassesFreq:=0;
inicializaNro_pendRelacs;
atualizaClasses_relacionamentos;
inicializaClasses_Freq;
atualizaNo_pendClasses;

clrscr;
writeln('Criando arquivo "'+ nome_arqSai + '"'+ '...');

{associa o arquivo de saida a uma variavel interna}
assign(arqSai,nome_arqSai);

{cria e abre o arquivo de saida}
rewrite(arqsai);

for w:=1 to tot_esq do {para cada esquema a ser gerado}
   begin
```

```
          qtdeClassesEsq:=0;
          {adiciona as classes ao esquema}
          usaClassesFreq;

          geraClasses;
          {Fim da geracao das classes}

          {gera atributos para cada classe}
          tamClasses_atribs:=0;
          for z:=1 to max_classes do
             begin
                tamClasses_atribs:=      geraAtributos(classes_esq[z],
tamClasses_atribs);
             end;
          {fim da geracao de atributos}

          {geracao do relacionamentos levando em consideracao as
classes do esquema e os relacionamentos usados}
          geraRelacionamentos;
          {fim da geracao dos relacs}

          {coloca dados do esquema no arquivo de saida}
          {transforma o numero do esquema para uma string com 4
posicoes}
          str(w, item);
          tam:= length(item);
          case tam of
            1: no_esq:='000';
            2: no_esq:='00';
            3: no_esq:='0';
            4: no_esq:='';
          end;
          insert(item,no_esq, 5-tam);
          {classes}
          for v:=1 to qtdeClassesEsq do
             if verClasses_esq(classes_esq[v], v-1) = false
             then begin
                    tam:=length(classes_esq[v]);
                    item:='               ';
                    insert(classes_esq[v],item, 16-tam);
                    reg:= no_esq + ' ' + item;
```

```
                writeln(arqSai, reg);
              end;

        {classes e atributos}
        for v:=1 to tamClasses_atribs do
           if verAtribs_esq (classes_atribs[v].atrib, v-1) = false
           then begin
                  tam:=length(classes_atribs[v].atrib);
                  item:='               ';
                  insert(classes_atribs[v].atrib,item, 16-tam);
                  reg:= no_esq + ' ' + item;
                  writeln(arqSai, reg);
                end;

        {relacionamentos}
        for v:=1 to tamRelacs_Esq do
           begin
              tam:=length(relacs_esq[v]);
              item:='               ';
              insert(relacs_esq[v],item, 16-tam);
              reg:= no_esq + ' ' + item;
              writeln(arqSai, reg);
           end;
        {fim do arquivo de saida}
     end;

     {fecha arquivo de saida}
     close(arqsai);

     writeln('Arquivo criado com sucesso!!');
     readln;
end.

{o algoritmo demora uma media de 7 min para gerar 1000 esquemas}
```

Anexo 3 Programa de Pós-processamento

O programa de pós-processamento automatiza a aplicação dos filtros apresentados na seção 5.2, tendo sido desenvolvido na linguagem Pascal.

A seguir é apresentado o código fonte do programa.

```
{Processa o arquivo de saida da mineracao de dados, deixando apenas as
regras cujo consequente

 e um relacionamento ou um pacote.

 Qdo o consequente e relacionamento, os antecedentes contem apenas
classes e/ou pacote ou apenas relacionamentos.

 Qdo o consequente e pacote, o antecedente so possui a classe a qual o
pacote pertence.


 Entrada: nome do arquivo de saida da mineracao.

 Saida: nome do arquivo de saida do pos-processamento


 Considera classes, pacotes, atributos e o esquema (E)


 FORMATO DE ENTRADA:

 pacote = P:

 classe = C

 pacote/classe = P: C

 classe/atributo = : C: a

 pacote/classe/atributo = P: C: a

 esquema = *E

 C1 relacionada a C2 = C1#C2#assoc }


program processa_saida;

uses crt;

const tamItem = 22; {tamanho da coluna de item do arquivo, contando
com os colchetes}

type

    string3 = array[1..3] of string;


    item = record

       tipo: integer; {0 - sem relac, 1 - com relac}

       componentes: string3;  {se for tipo = 0, recebe itensNaoRelac}

    end;


    ant = record

       n: integer; {nro. de itens que compoem o antecedente}
```

```
       componentes: array[1..20] of item;  {cada item separadamente}
    end;


var i: integer;
    iniCol, fimCol,fimInf, iniInf, iniInfCopy: integer;
    cont: longint;
    arqEnt, arqSai: text;
    linha, arquivoE,arquivoS, contStr, copyLinha: string;
    conseq: string;        {guarda o consequente da regra}
    antec: string;         {guarda o antecedente da regra}
    itensNaoRelac: string3; {contem, respectivamente, os nomes do
pacote, classe e atributo de um item}
    itensRelac: string3;     {contem, respectivamente, os nomes das
classes e o tipo do relacionamento}
    consequente: item;      {guarda o consequente da regra, ja com os
itens separados}
    antecedente: ant;       {guarda cada item do antecedente da regra
separadamente}
    escreve: boolean;       {variavel de controle. O default eh true,
posi sempre escreve a linha na saida a menos que
                            nao satisfaca a condicao de algum filtro}


{recebe o item "item"(sem colchetes) composto por um relacionamento
entre duas classes
 e separa na variavel global "itensRelac"}
procedure separaItemRelac (item: string);
var sepCls: integer;
    sepTpRelac: integer;
    i: integer;


begin
   for i:= 1 to 3 do
      itensRelac[i]:= '';


   i:= pos(' ', item);
   while i = 1 do
     begin
       item:= copy(item,2, length(item));
       i:= pos(' ', item);
     end;


   {separa as classes}
```

```
   sepCls := pos('#', item);
   itensRelac[1]:= copy(item, 1, sepCls-1);
   item:= copy(item, sepCls+1, length(item));
   sepTpRelac := pos('#', item);
   itensRelac[2]:= copy(item, 1, sepTpRelac-1);
   item:= copy(item, sepTpRelac+1, length(item));
   itensRelac[3]:= item;
end;

{recebe o item "item"(sem colchetes) composto por pacote, classe e/ou
atributo e os separa na variavel global "itensNaoRelac"
Se fo o esquema (*E), ele ficar'a na posi'c~ao da classe na variavel
'itensNaoRelac'}
procedure separaItemNaoRelac (item: string);
var sepPacCl: integer;
    sepClAt: integer;
    i: integer;

begin
   for i:= 1 to 3 do
      itensNaoRelac[i]:= '';

   i:= pos(' ', item);
   while i = 1 do
     begin
       item:= copy(item,2, length(item));
       i:= pos(' ', item);
     end;

   {separa pacote de classe}
   sepPacCl := pos(':', item);
   if sepPacCl > 1      {tem info de pacote}
   then itensNaoRelac[1]:= copy(item, 1, sepPacCl - 1)
   else if sepPacCl = 0 {so tem classe}
        then begin
                itensNaoRelac[2]:= item;
                exit;
             end;

   item := copy(item, sepPacCl + 2, length(item));
```

```
   {separa classe de atributo}
   sepClAt := pos(':', item);
   if sepClAt > 0      {tem info de atributo}
   then begin {tem info de atributo}
           itensNaoRelac[2]:= copy(item, 1, sepClAt - 1);
           item := copy(item, sepClAt + 2, length(item));
           itensNaoRelac[3]:= copy(item, 1, length(item));
        end
   else itensNaoRelac[2]:= copy(item, 1, length(item));
end;


{separa o antecedente do consequente da regra. Coloca o item do consequente, ja separado na var global 'consequente' e
os itens do antecedente, tb ja separados, na var global 'antecedente'}
procedure separaAntecConseq;
begin
   {pega o consequente da regra}
   conseq:=  (copy(linha,(fimInf+3),  length(linha)));  {conseq  sem colchete inicial}
   conseq  :=  copy(conseq,  1  ,  length(conseq)-1);        {conseq sem colchete final}
   {le a posicao do inicio da seta que indica a inferencia}
   iniInf:= Pos('=', linha);
   copyLinha:= linha;
   iniCol:= Pos('[', copyLinha);
   fimCol:= Pos(']', copyLinha);
   iniInfCopy:= iniInf;

   {se for relacionamento no consequente}
   if pos('#', conseq) > 0
   then begin
          separaItemRelac (conseq);
          consequente.tipo:= 1;
          consequente.componentes:=  itensRelac;  {guarda  itens  do conseq ja separados}
        end
   else begin {nao e relacionamento}
           separaItemNaoRelac (conseq);
           consequente.tipo:= 0;
           consequente.componentes:= itensNaoRelac; {guarda itens do conseq ja separados}
        end;
```

```
   antecedente.n := 0;
   while iniCol < iniInfCopy do  {esta pegando o antecedente}
      begin
         antecedente.n := antecedente.n + 1;
         antec:= copy(copyLinha,(iniCol+1), tamItem-2);
         if pos('#',antec) = 0
         then begin
                 separaItemNaoRelac(antec);
                 antecedente.componentes[antecedente.n].tipo:= 0;
                 antecedente.componentes[antecedente.n].componentes:=
itensNaoRelac;
              end
         else begin
                 separaItemRelac(antec);
                 antecedente.componentes[antecedente.n].tipo:= 1;
                 antecedente.componentes[antecedente.n].componentes:=
itensRelac;
              end;

         copyLinha:= copy(copyLinha,(fimCol+1), (iniInf-1));
         iniCol:= Pos('[', copyLinha);
         fimCol:= Pos(']', copyLinha);
         iniInfCopy:= Pos('=', copyLinha);
      end;

end;

{ INI NOVO CODIGO }

{ver tipo de item naoRelac.
 Entrada: item do tipo naoRelac
 Saida: 1 - so pacote
        2 - so classe
        3 - pacote e classe
        4 - classe e atributo
        5 - pacote, classe e atributo
        6 - esquema}
function tipoItemNaoRelac(itemNaoRelac: string3): integer;
begin
```

```
   tipoItemNaoRelac := 0;

   if itemNaoRelac[2] = '*E'
   then begin
           tipoItemNaoRelac := 6;
           exit;
        end;

   if itemNaoRelac[1] <> '' {tem pacote}

   then if itemNaoRelac[2] = ''  {nao tem classe}
        then begin
                tipoItemNaoRelac := 1;
                exit;
             end
        else if itemNaoRelac[3] <> '' {tem atributo}
             then begin
                     tipoItemNaoRelac := 5;
                     exit;
                  end
             else begin {so tem pacote e classe}
                     tipoItemNaoRelac := 3;
                     exit;
                  end
   else {nao tem pacote}
        if itemNaoRelac[2] <> ''  {tem classe}
        then if itemNaoRelac[3] <> '' {tem atributo}
             then begin
                     tipoItemNaoRelac := 4;
                     exit;
                  end
             else begin {so classe}
                     tipoItemNaoRelac := 2;
                     exit;
                  end;
end;


{Verifica esquemas no consequente.
 Retorna TRUE nao tiver esquema no consequente e
```

```
 FALSE se existir esquema no consequente}
function filtro1_0: boolean;
begin

   filtro1_0 := true;

   if           (consequente.tipo           =           0)           and
(tipoItemNaorelac(consequente.componentes) = 6)
   then filtro1_0 := false;

end;

{Se o antecedente tiver *E, deve ser so ele e o consequente tem que
ser classe ou pacote.
 Retorna TRUE se satisfizer as condicoes e
 FALSE se nao satisfizer}
function filtro1_1: boolean;
var i: integer;
    existeE: boolean;

begin
  filtro1_1 := true;
  existeE := false;

  for i:= 1 to antecedente.n do
     begin
        if      (antecedente.componentes[i].tipo      =      0)      and
(tipoItemNaorelac(antecedente.componentes[i].componentes) = 6)
        then existeE:= true;
     end;

  if (antecedente.n > 1)
  then begin
         if existeE = true
         then filtro1_1 := false;
       end
  else if existeE = true
       then begin {verifica se o consequente e so classe ou so pacote}
              if consequente.tipo = 1
              then filtro1_1 := false {consequente e relacionamento}
              else if (tipoItemNaoRelac(consequente.componentes) = 1)
or (tipoItemNaoRelac(consequente.componentes) = 2)
```

```
                    then filtro1_1 := true
                    else filtro1_1 := false;
             end;
end;

{Se o consequente nao eh relacionamento, o antecedente nao deve
possuir relacionamento.
 TRUE se satisfaz}
function filtro2_0: boolean;
var i:integer;
begin
   filtro2_0 := true;

   if consequente.tipo = 0
   then begin
           for i:= 1 to antecedente.n do
              begin
                 if antecedente.componentes[i].tipo = 1 {eh um relac}
                 then begin
                         filtro2_0 := false;
                         exit;
                      end;
              end;
        end;
end;

{consequente nao eh relac. Se tem mais de um item no antecedente e
todos os itens sao classe, descarta.
 So vale qdo eh mais de um item, pois uma unica classe, pode implicar
em classe/atributo.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro2_1: boolean;
var i, cont: integer;

begin
   filtro2_1 := true;
   cont:= 0;

   if (consequente.tipo = 0) and (antecedente.n > 1)
   then begin
```

```
           for i:= 1 to antecedente.n do
              begin
                 if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 2 {item eh
so classe}
                 then cont:= cont + 1;
              end;

           if cont = antecedente.n
           then filtro2_1 := false;
        end;
end;

{consequente nao eh relac. Se todos os itens do antecedente sao
pacotes, descarta.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro2_2: boolean;
var i, cont: integer;

begin
   filtro2_2 := true;
   cont:= 0;

   if (consequente.tipo = 0)
   then begin
           for i:= 1 to antecedente.n do
              begin
                 if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 1 {item eh
so pacote}
                 then cont:= cont + 1;
              end;

           if cont = antecedente.n
           then filtro2_2 := false;
        end;
end;

{apenas classe no consequente, antecedente so pode ser *E.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
```

```
 com relac}
function filtro20_0: boolean;
begin
   filtro20_0 := true;

   if consequente.tipo = 0
   then if tipoItemNaorelac(consequente.componentes) = 2
        then begin
                if antecedente.n > 1
                then filtro20_0 := false {mais de um item, ja nao pode
ser so *E}
                else begin
                       if antecedente.componentes[1].componentes[2] <>
'*E'
                       then filtro20_0 := false;
                     end;
             end;
end;


{apenas pacote no consequente, antecedente so pode ser *E.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro21_0: boolean;
begin
   filtro21_0 := true;

   if consequente.tipo = 0
   then if tipoItemNaorelac(consequente.componentes) = 1
        then begin
                if antecedente.n > 1
                then filtro21_0 := false {mais de um item, ja nao pode
ser so *E}
                else begin
                       if antecedente.componentes[1].componentes[2] <>
'*E'
                       then filtro21_0 := false;
                     end;
             end;
end;
```

```
{consequente composto por classe/atributo, antecedente deve possuir
todos os itens com a classe
 igual a do consequente.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro22_0: boolean;
var i: integer;
begin
   filtro22_0 := true;

   if consequente.tipo = 0
   then if tipoItemNaorelac(consequente.componentes) = 4
        then begin
                for i:= 1 to antecedente.n do
                   begin
                       if antecedente.componentes[1].componentes[2] <>
consequente.componentes[2]
                       then begin
                               filtro22_0 := false;
                               exit;
                            end;

                   end;
            end;
end;

{consequente composto por classe/atributo, a classe nao deve estar
replicada no antecedente, se nao possuir info adicional
 DEVE SER USADO SEMPRE APOS O FILTRO22_0, para eliminar regras com
itens no antecedente com classe diferente da do consequente}
function filtro22_1: boolean;
var i: integer;
begin
   filtro22_1 := true;

   if consequente.tipo = 0
   then if (tipoItemNaorelac(consequente.componentes) = 4) and
(antecedente.n > 1)
        then begin
                for i:= 1 to antecedente.n do
                   begin
```

```
                    if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 2
                    then begin   {tem um item que eh so a classe do
consequente, se o antecedente tem mais de um item e
                                 todos eles devem ter a classe
igual a do consequente, pode-se afirmar que existe
                                 replicacao}
                            filtro22_1 := false;
                            exit;
                         end;
                 end;
            end;
end;


{consequente composto por pacote/classe, deve possuir no minimo dois
itens no antecedente e
 nao deve possuir no antecedente itens com pacote/classe/atributo.
 DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro23_0: boolean;
var i: integer;
begin
   filtro23_0 := true;


   if consequente.tipo = 0
   then if (tipoItemNaorelac(consequente.componentes) = 3)
        then if antecedente.n < 2
             then filtro23_0 := false
             else begin {verifica se existe pacote/classe/atributo no
antecedente}
                  for i:= 1 to antecedente.n do
                     begin
                        if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 5
                        then begin
                                filtro23_0 := false;
                                exit;
                             end;
                     end;
                end;
end;
```

```
{consequente composto por pacote/classe, deve possuir no antecedente, pelo menos dois itens, um com a info do pacote
 outro com a info da classe do consequente.
 DEVE SER USADO SEMPRE APOS O FILTRO23_0, para eliminar regras com menos de dois itens no antecedente e com
 pacote/classe/atributo no antecedente}
function filtro23_1: boolean;
var i: integer;
    entrouPac, pacotes, classes: boolean;
begin
   filtro23_1 := true;
   pacotes := false;
   classes:= false;

   if consequente.tipo = 0
   then if (tipoItemNaorelac(consequente.componentes) = 3)
        then begin {verifica se a classe e o pacote do consequente estao no antecedente}
                for i:= 1 to antecedente.n do
                   begin
                      entrouPac:= false; {para evitar que considere pacote e classe do mesmo item}
                      if (pacotes = false) and (antecedente.componentes[i].componentes[1] = consequente.componentes[1])
                      then begin
                             pacotes:= true; {o pacote do consequente existe no antecedente}
                             entrouPac:= true;
                           end;

                      if (entrouPac = false) and (antecedente.componentes[i].componentes[2] = consequente.componentes[2])
                      then classes:= true; {a classe do consequente existe no antecedente}
                   end;

                if (pacotes = false) or (classes = false)
                then filtro23_1 := false;
             end;
end;
```

```
{consequente composto por pacote/classe, o antecedente so pode possuir
itens com pacote igual ao do consequente
 ou classe igual a do consequente
 DEVE SER USADO SEMPRE APOS O FILTRO23_1, para eliminar regras com
consequente diferente de relac e antecedente
 com relac}
function filtro23_2: boolean;
var i: integer;
begin
   filtro23_2 := true;

   if consequente.tipo = 0
   then if (tipoItemNaorelac(consequente.componentes) = 3)
        then begin
               for i:= 1 to antecedente.n do
                  begin
                     if  antecedente.componentes[i].componentes[1]  <>
'' {existe info de pacote}
                     then if antecedente.componentes[i].componentes[1]
<> consequente.componentes[1] {pac antec <> do conseq}
                          then begin
                                  filtro23_2:= false;
                                  exit;
                               end;

                     if  antecedente.componentes[i].componentes[2]  <>
'' {existe info de classe}
                     then if antecedente.componentes[i].componentes[2]
<> consequente.componentes[2] {class antec <> do conseq}
                          then begin
                                  filtro23_2:= false;
                                  exit;
                               end;
                  end;
             end;
end;

{{{{}
{{{Pacote/classe no consequente, falta verificar info duplicada no
antecedente}
{{{{}
```

```
{consequente com pacote/classe/atributo, se tiver info soh de classe, soh de
 pacote ou soh de classe/atrib no antecedente, descarta, pois ela eh replicada,
 ja que deve existir, pelo menos pacote/classe ou pacote/classe/atrib.
 ou seja, soh permite no antecedente itens sejam todos pacote/classe ou
 pacote/classe/atrib.
 Se tiver apenas um item no antecedente, a classe tem que ser igual a do
 consequente.
  DEVE SER USADO SEMPRE APOS O FILTRO2_0, para eliminar regras com consequente diferente de relac e antecedente
 com relac}
function filtro24_0: boolean;
var i: integer;
    tipoConseqNaoRelac, TipoAntecNaoRelac: integer;


begin
   filtro24_0 := true;


   if (consequente.tipo = 0)
   then begin
           tipoConseqNaoRelac                                          :=
tipoItemNaorelac(consequente.componentes);
           if     tipoConseqNaorelac     =     5     {o     item     eh
pacote/classe/atributo}
           then begin
                   if antecedente.n = 1
                   then begin
                           if
antecedente.componentes[1].componentes[2]                              <>
consequente.componentes[2]
                           then begin
                                   filtro24_0 := false;
                                   exit;
                                end;
                        end
                   else begin
                          for i:= 1 to antecedente.n do
                             begin
                                tipoAntecNaoRelac:=
tipoItemNaoRelac(antecedente.componentes[i].componentes);
```

```
                                    if    (tipoAntecNaoRelac    =    1)    or
(tipoAntecNaoRelac = 2) or (tipoAntecNaoRelac = 4)  {so pacote ou

classe ou classe/atrib}
                                    then begin
                                            filtro24_0 := false;
                                            exit;
                                        end;
                                end;
                            end;
                    end;


        end;
end;


{consequente com pacote/classe/atributo, soh permite no antecedente
itens cujo
 pacote seja igual ao do conseq
 DEVE SER USADO SEMPRE APOS O FILTRO24_0, para eliminar regras com
antecedente diferente de pacote/classe ou
 pacote/classe/atrib}
function filtro24_1: boolean;
var i: integer;


begin
   filtro24_1 := true;


   if (consequente.tipo = 0)
   then if tipoItemNaoRelac(consequente.componentes) = 5
        then begin
                for i:= 1 to antecedente.n do
                   begin
                      if  antecedente.componentes[i].componentes[1]  <>
consequente.componentes[1]
                      then begin
                              filtro24_1 := false;
                              exit;
                           end;
                   end;


             end;
end;
```

```
{qdo e relacionamento no consequente, tem que ter, pelo menos, as duas
classes
 no antecedente}
function filtro3_0: boolean;
var classe1, classe2: boolean;
    i: integer;
begin
    filtro3_0:= true;
    classe1:= false;
    classe2:= false;

    if consequente.tipo = 0     {nao eh relac no consequente}
    then exit;

    for i:= 1 to antecedente.n do
       begin
          if antecedente.componentes[i].tipo = 0 {o item nao eh relac}
          then begin
                   if    antecedente.componentes[i].componentes[2]    =
consequente.componentes[1]
                   then classe1:= true     {igual a primeira classe do
relac}
                   else  if  antecedente.componentes[i].componentes[2]  =
consequente.componentes[2]
                        then classe2:= true; {igual a classe 2}
               end
          else begin   {eh relac}
                   if     antecedente.componentes[i].componentes[1]     =
consequente.componentes[1]
                   then classe1 := true
                   else  if  antecedente.componentes[i].componentes[1]  =
consequente.componentes[2]
                        then classe2 := true;

                   if     antecedente.componentes[i].componentes[2]     =
consequente.componentes[1]
                   then classe1 := true
                   else  if  antecedente.componentes[i].componentes[2]  =
consequente.componentes[2]
                        then classe2 := true;
               end;
       end;
```

```
    if (classe1<>true) or (classe2<>true)
    then filtro3_0 := false;
end;

{qdo eh relacionamento no consequente, nao pode ter info soh de pacote
 no antecedente}
function filtro3_1: boolean;
var i: integer;
begin
    filtro3_1:= true;

    if consequente.tipo = 0     {nao eh relac no consequente}
    then exit;

    for i:= 1 to antecedente.n do
       begin
          if antecedente.componentes[i].tipo = 0 {o item nao eh relac}
          then begin
                  if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 1 {tem item
no antecedente que eh soh pacote}
                  then begin
                          filtro3_1:= false;
                          exit;
                       end;
               end;
       end;
end;

{qdo eh relacionamento no consequente, se tem info soh de classe no
antecedente,
 ela deve ser igual a uma das classes do consequente, pois classes
diferentes
 devem pertencer a relacionamento ou apresentar.}
function filtro3_2: boolean;
var i: integer;
begin
    filtro3_2:= true;

    if consequente.tipo = 0     {nao eh relac no consequente}
    then exit;
```

```
    for i:= 1 to antecedente.n do
       begin
          if antecedente.componentes[i].tipo = 0 {o item nao eh relac}
          then begin
                  if
tipoItemNaoRelac(antecedente.componentes[i].componentes) = 2 {tem item
no antecedente que eh soh classe}
                  then begin
                          if
(antecedente.componentes[i].componentes[2])                          <>
(consequente.componentes[1])
                          then                                         if
(antecedente.componentes[i].componentes[2])                          <>
(consequente.componentes[2])
                               then begin
                                       filtro3_2:= false;
                                       exit;
                                    end;
                      end;
              end;
       end;
end;

{   IMPLEMENTAR
verifica se a classe CLASSE pertence a algum relacionamento existente
no
 antecedente. Retorna TRUE caso positivo e FALSE, se nao existir relac
no
 antecedente ou se a classe nao pertence a um relacionamento do antec}
function existeClasseEmRelacAntec(classe: string): boolean;
var i: integer;
    ok, temRelac : boolean;
begin
   existeClasseEmRelacAntec:= true;
   temRelac:= false;

   for i:= 1 to antecedente.n do
      if antecedente.componentes[i].tipo = 1 {eh relacionamento}
      then begin
              temRelac:= true;
              exit;
           end;
```

```
   if temRelac = false {nao tem nenhum relac no antecedente}
   then existeClasseEmRelacAntec := false
   else begin
           for i:= 1 to antecedente.n do
              begin
                 if antecedente.componentes[i].tipo = 1 {eh relac}
                 then begin {verifica se a classe CLASSE eh igual a
uma das classes desse relac}
                         if                  classe                  =
antecedente.componentes[i].componentes[1]
                         then begin
                                 ok:= true;
                                 exit;
                              end
                         else                if        classe        =
antecedente.componentes[i].componentes[2]
                               then begin
                                      ok:= true;
                                      exit;
                                    end
                     end;
              end;
        end;

   if ok = false
   then existeClasseEmRelacAntec := false;
end;


{qdo eh relacionamento no consequente, se tem info de pacote ou
atributo no
 antecedente, a classe relacionada deve ser igual a uma das classes do
 consequente ou a uma classe que esteja em um relacionamento no
antecedente
 DEVE SER USADO APOS FILTRO3_1, para evitar info soh de pacote no
antec}
function filtro3_3: boolean;
var i: integer;
    tipoAntecNaoRelac: integer;

begin
    filtro3_3:= true;
```

```
    if consequente.tipo = 0     {nao eh relac no consequente}
    then exit;

    for i:= 1 to antecedente.n do
       begin
          if antecedente.componentes[i].tipo = 0 {o item nao eh relac}
          then begin
                  tipoAntecNaoRelac                                    :=
tipoItemNaoRelac(antecedente.componentes[i].componentes);
                  if (tipoAntecNaoRelac = 3) or (tipoAntecNaoRelac =
4) or (tipoAntecNaoRelac = 5)
                       {tem item no antecedente que eh pacote/classe,
classe/atrib ou pacote/classe/atrib}
                  then begin
                          if
(antecedente.componentes[i].componentes[2])                            <>
(consequente.componentes[1])
                          then                                         if
(antecedente.componentes[i].componentes[2])                            <>
(consequente.componentes[2])
                               then begin
                                       {a  classe  nao  eh  nenhuma  das
duas do conse}
                                       if
existeClasseEmRelacAntec(antecedente.componentes[i].componentes[2])   =
false
                                       then begin
                                               {a    classe    tb    nao
pertence a nenhum relac do antec}
                                               filtro3_3:= false;
                                               exit;
                                          end;
                                  end;
                       end;
              end;
       end;
end;

{rever...estsh excluindo indevidamente regra do tipo C+P:C:A -> assoc}

{qdo  eh  relacionamento  no  consequente,  verifica  se  nao  tem  info
duplicada de
 classe.}
```

```
function filtro3_4: boolean;
var i, j, tipoAntecNaoRelac, tipoAntec2NaoRelac: integer;
begin
   filtro3_4 := true;


   for i:= 1 to antecedente.n do
       begin
          if (antecedente.componentes[i].tipo) = 0
          then begin
                   tipoAntecNaoRelac:=
tipoItemNaoRelac(antecedente.componentes[i].componentes);
                   if (tipoAntecNaoRelac = 2) or (tipoAntecNaoRelac =
5){soh classe ou pac/classe/atrib}
                   then begin
                          {nao pode ter a mesma classe em pac/classe;
classe/atrib ou pac/classe/atrib}
                          for j:= 1 to antecedente.n do
                              begin
                                 if antecedente.componentes[j].tipo = 0
                                 then begin
                                         tipoAntec2NaoRelac:=
tipoItemNaoRelac(antecedente.componentes[j].componentes);
                                         if (tipoAntec2NaoRelac = 3) or
(tipoAntec2NaoRelac = 4) or (tipoAntec2NaoRelac = 5)
                                    then                                  if
antecedente.componentes[i].componentes[2]                                  =
antecedente.componentes[j].componentes[2]
                                                then begin
                                                        filtro3_4:=
false;
                                                        exit;
                                                    end;
                                        end;
                                 end;
                          end
                   else if (tipoAntecNaoRelac = 3)  {pacote/classe}
                          then begin
                                  {nao  pode  ter  a  mesma  classe  em
classe/atrib ou pac/classe/atrib}
                                 for j:= 1 to antecedente.n do
                                     begin
                                        if
antecedente.componentes[j].tipo = 0
                                        then begin
```

```
                                                tipoAntec2NaoRelac:=
tipoItemNaoRelac(antecedente.componentes[j].componentes);
                                                if  (tipoAntec2NaoRelac  =
2) or (tipoAntec2NaoRelac = 5)
                                    then                                    if
antecedente.componentes[i].componentes[2]                                    =
antecedente.componentes[j].componentes[2]
                                                        then begin
                                                                filtro3_4:=
false;
                                                                exit;
                                                        end;
                                                end;
                                        end;
                                end
                        else     if     (tipoAntecNaoRelac     =     4)
{pacote/classe}
                                then begin
                                        {nao  pode  ter  a  mesma  classe  em
classe/atrib ou pac/classe/atrib}
                                        for j:= 1 to antecedente.n do
                                            begin
                                               if
antecedente.componentes[j].tipo = 0
                                               then begin
                                                     tipoAntec2NaoRelac:=
tipoItemNaoRelac(antecedente.componentes[j].componentes);
                                    if    (tipoAntec2NaoRelac    =    2)    or
(tipoAntec2NaoRelac = 5) or (tipoAntec2NaoRelac = 5)
                                    then                                    if
antecedente.componentes[i].componentes[2]                                    =
antecedente.componentes[j].componentes[2]
                                                        then begin

filtro3_4:= false;
                                                                exit;
                                                        end;
                                                end;
                                        end;
                                end;
                        end;
                end;
end;

{ FIM NOVO CODIGO }
```

```
{programa principal}
begin

   clrscr;
 {  writeln ('Informe o nome do arquivo a ser processado:');
   readln(arquivoE);}
   writeln('Processando...');

   arquivoE:='C:\carol\DISS_1R.txt';

   assign(arqEnt, arquivoE);
   reset(arqEnt);

   arquivoS:='C:\carol\DISS_1RP.txt';
   assign(arqSai, arquivoS);
   rewrite(arqSai);

   writeln(arqSai, 'Processado pelo programa "POSPROc6.PAS".');
   writeln(arqSai);

   cont:=0; {teste}

   while not eof(arqEnt) do
      begin
        readln(arqEnt,linha);

        {le a posicao do fim da seta que indica a inferencia}
        fimInf:= Pos('>', linha);

        if fimInf > 0
        then {so executa se for realmente uma regra. Ou seja, a linha
possui a seta}
             begin
                separaAntecConseq;

                 {o default eh escrever a regra. Se nao satisfizer
alguma condicao de filtro, descarta}
                 escreve:= true;
```

```
if filtro1_0 = false
then escreve:= false;
if filtro1_1 = false
then escreve:= false;

if filtro2_0 = false
then escreve:= false;
if filtro2_1 = false
then escreve:= false;
if filtro2_2 = false
then escreve:= false;
if filtro20_0 = false
then escreve:= false;
if filtro21_0 = false
then escreve:= false;
if filtro22_0 = false
then escreve:= false;
     if filtro22_1 = false
     then escreve:= false;
if filtro23_0 = false
then escreve:= false;
     if filtro23_1 = false
     then escreve:= false;
          if filtro23_2 = false
          then escreve:= false;
if filtro24_0 = false
then escreve:= false;
     if filtro24_1 = false
     then escreve:= false;

if filtro3_0 = false
then escreve:= false;
if filtro3_1 = false
then escreve:= false;
if filtro3_2 = false
then escreve:= false;
     if filtro3_3 = false
     then escreve:= false;
     if filtro3_4 = false
     then escreve:= false;
```

```
                if escreve = true
                then begin
                        writeln(arqSai, linha);
                        cont:= cont+1;
                     end;
            end;
      end;

   Str(cont, contstr);

   {grava no arquivo de saida a qtde total de regras apos o filtro}
   writeln(arqSai);
   writeln(arqSai, 'Total de linhas no arquivo de saida: '+ contStr);

   close(arqEnt);
   close(arqsai);

   writeln('Total de linhas no arquivo de saida: '+ contStr);
   readln;
end.
```