Pós-Graduação em Ciência da Computação

SKDQL

Uma Linguagem Declarativa de Especificação de Consultas e Processos para Descoberta de Conhecimento em Bancos de Dados e sua Implementação

Por

Marcelino Pereira dos Santos Silva

Dissertação de Mestrado

Orientador

Prof. Dr. Jacques Pierre Louis Robin

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

Recife, 26 de fevereiro de 2002

Universidade Federal de Pernambuco

Centro de Informática

Marcelino Pereira dos Santos Silva

SKDQL
Uma Linguagem Declarativa de Especificação de Consultas e Processos para Descoberta de Conhecimento em Bancos de Dados e sua Implementação

Dissertação apresentada ao Centro de Informática da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação, sob a orientação do Prof. Dr. Jacques Robin.

Recife, 26 de fevereiro de 2002.

681.3.016

SILVA, M. P. dos S.

SKDQL: uma linguagem declarativa de especificação de consultas e processos para descoberta de conhecimento em bancos de dados e sua implementação. M. P. dos S. Silva. Recife: UFPE, 2002.

103 p.

1. Bancos de dados. 2. Mineração de dados 3. Ciência da Computação. I. Título

Agradecimentos

Neste momento, palavras possuem pouca expressividade para externar minha gratidão ao Deus Todo-Poderoso, o Senhor dos Exércitos, que me concede mais esta vitória. A Ele toda honra, toda glória, todo louvor.

Agradeço à minha família pelo apoio irrestrito, em todos os momentos. Obrigado Leni pela força, compreensão, conselho e carinho; obrigado Daniel e Davi pelos olhos sinceros, pelo abraço amigo e pela alegria de ver-lhes me chamando de papai – o simples ato de pensar em vocês me inspira e me traz felicidade. Obrigado Dona Augusta, minha mãe forte, querida e abençoada, que com grande esforço testemunhou esta vitória. Agradeço a Deus por Seu Virgílio, que mesmo tendo partido muito cedo ensinou-me tudo que um grande pai pode dar e ser: amor, dignidade, respeito. Obrigado Marcelo, parceiro nos sonhos e lutas da vida, pelo incentivo e apoio que apenas um irmão de verdade pode dar.

Minha gratidão ao Professor Jacques Robin pela competente orientação, pelos conhecimentos transmitidos, pela experiência repassada e pelo apoio decisivo na consolidação deste trabalho.

Agradeço também aos membros da Banca de Defesa, Professor Fernando Fonseca e Professor Marcus Sampaio, pela atenção, apoio e grande colaboração nesta fase de minha formação.

Minha gratidão à direção e aos colegas da Universidade Estadual do Rio Grande do Norte pelo suporte, incentivo e voto de confiança neste desafio.

Sou muito grato àqueles que fazem o CIn: docentes, que com competência transmitiram conteúdos e lições preciosas; obrigado aos funcionários que cotidianamente me apoiaram de forma valorosa, em especial Lilia, Neide e Nadja. Agradeço ainda aos colegas de curso com quem compartilhei experiências, alegrias, dificuldades e realizações, que muito colaboraram com meu trabalho, em especial Alexandre Damasceno, Erivan Alves, Fábio Massoud, João Batista, Rodrigo Galvão, Vitor Campos: grandes amigos e profissionais.

Obrigado Bezerra e Willame. Com vocês reaprendi o significado da palavra amigo.

Obrigado a todos que de forma direta ou indireta contribuiram com este trabalho.

Que Deus abençoe cada um de vocês.

Resumo

As ferramentas e técnicas empregadas para análise automática e inteligente dos imensos repositórios de dados de indústrias, governos, corporações e institutos científicos são os objetos tratados pelo campo emergente da Descoberta de Conhecimento em Bancos de Dados (Knowledge Discovery in Databases - KDD). No contexto do MATRIKS, um framework para KDD, SKDQL (Structured Knowledge Discovery Query Language) é a proposta de uma linguagem de consulta estruturada para KDD, seguindo os padrões de SQL dentro de uma arquitetura aberta e extensível, suportando a heterogeneidade, iteratividade e interatividade dos processos de KDD, com recursos para acesso, limpeza, transformação, derivação e mineração de dados, bem como manipulação de conhecimento.

Abstract

Tools and techniques employed for automatic and smart analysis of huge data repositories of industries, governments, corporations and scientific institutes are the subjects dealt by the emerging field of Knowledge Discovery in Databases (KDD). In MATRIKS context, a framework for KDD, SKDQL (Structured Knowledge Discovery Query Language) is the proposal of a structured query language for KDD, following SQL patterns within an open and extensible architecture, supporting heterogeneity, interaction and increment of KDD process, with resources for accessing, cleaning, transforming, deriving and mining data, beyond knowledge manipulation.

Índice

Índice de Figuras

1 Introdução e Motivação

Problemática Geral: Descoberta de Conhecimento em Bancos de Dados

As áreas governamentais, corporativas e científicas têm promovido um crescimento explosivo em seus bancos de dados, superando em muito a usual capacidade de interpretar e examinar estes dados, gerando a necessidade de novas ferramentas e técnicas para análise automática e inteligente de bancos de dados [FAY1996].

Nos diferentes segmentos da sociedade, as instituições têm buscado na tecnologia recursos que agreguem valor aos seus negócios, seja agilizando operações, suportando ambientes ou viabilizando inovações. Diariamente, pessoas e instituições disponibilizam dados oriundos de tarefas cotidianas a estas plataformas tecnológicas através de simples atividades como compras no supermercado do bairro ou operações bancárias. Os sistemas de computação participam da vida das pessoas de forma cada vez mais próxima e constante. Não obstante, institutos científicos, indústrias, corporações e governos acumulam volumes gigantescos de dados, impulsionados também pela versatilidade e alcance proporcionados pela Internet.

Esta ampla disponibilidade de imensas bases de dados, aliada à eminente necessidade de transformar tais dados em informação e conhecimento úteis para o suporte à decisão [CHA1998], têm demandado investimentos consideráveis da comunidade científica e da indústria de software. A informação e o conhecimento obtidos podem ser utilizados para diversas aplicações, que vão do gerenciamento de negócios, controle de produção e análise de mercado ao projeto de engenharia e exploração científica [HAN2001].

As ferramentas e técnicas empregadas para análise automática e inteligente destes imensos repositórios são os objetos tratados pelo campo emergente da descoberta de conhecimento em bancos de dados (DCBD), da expressão em inglês Knowledge Discovery in Databases (KDD).

O Processo de Descoberta de Conhecimento em Bancos de Dados

Descoberta de conhecimento em bancos de dados é o processo não trivial de identificar em dados padrões (tendências, comportamentos) que sejam válidos (possuam grau

de certeza aceitável), novos (previamente desconhecidos), potencialmente úteis (capazes de indicar medidas práticas) e compreensivos (representação simples, intuitiva), visando melhorar o entendimento de um problema ou um procedimento de tomada de decisão [FAY1996].

O processo de KDD é interativo, iterativo, cognitivo e exploratório, envolvendo passos como: definição do tipo de conhecimento a descobrir, seleção de dados alvo, pré-processamento, transformação, mineração destes dados, subseqüente interpretação de padrões e implantação do conhecimento descoberto. A cada etapa, passos anteriormente realizados podem ser retomados com base em observações e descobertas realizadas até aquele momento, visando solucionar problemas encontrados ou aumentar a qualidade dos resultados.

Este processo pode ser aplicado nos mais diversos domínios: comércio e indústria (vendas de supermercados, produção industrial, marketing); finanças (operações de bancos, financeiras, seguradoras); governo (estatísticas de saúde, levantamentos demográficos); tecnologia (logs de servidores, sistemas, simuladores); descoberta científica (genética, astronomia, geologia, medicina), dentre várias outras aplicações.

Gargalos do Processo de DCBD

Ambientes de KDD existentes foram projetados como sistemas monolíticos e isolados em oposição às principais tendências da engenharia de software moderna: sistemas distribuídos, abertos, onde cada funcionalidade é encapsulada em um componente reutilizável, tendo estes componentes a comunicação efetuada através de middleware no contexto de um framework [BUE1998]. Outra característica comum aos sistemas de KDD é que eles oferecem um conjunto limitado e pequeno de métodos para as diferentes tarefas do processo de descoberta de conhecimento. Eles são produtos fechados do ponto de vista do usuário, impossibilitando para este o desenvolvimento e a integração de novos métodos de mineração de dados que melhor se apliquem ao problema em mãos.

Outro ponto relevante é a funcionalidade destes sistemas no que diz respeito ao gerenciamento de conhecimento, que é um processo sistêmico e devidamente especificado para aquisição, organização e comunicação tácita e explícita do conhecimento de modo que outros possam fazer uso deste para atingir objetivos pré-definidos. Descoberta, captura, integração e transmissão do conhecimento configuram-se em grandes desafios deste contexto.

Estado da Arte

DMQL (Data Mining Query Language) [HAN1996] é uma proposta que permite a mineração ad hoc de vários tipos de conhecimento a partir de bancos de dados transacionais e data warehouses em vários níveis de abstração. A linguagem adota uma sintaxe semelhante a SQL. Entretanto não se encontra implementada, carecendo a sua especificação de métodos para pré-processamento de dados, além de contar com um conjunto muito limitado e fechado de algoritmos de mineração.

A proposta da Microsoft - OLE DB for Data Mining (OLE DB for DM) [MIC2000] - é uma especificação em direção à padronização das primitivas de linguagem de mineração de dados, visando tornar-se um padrão de indústria. Ela é desenhada para permitir aplicações de mineração de dados do cliente, consumindo serviços de uma variedade de provedores ou pacotes de software de mineração de dados. A especificação possui poucos métodos de mineração embutidos, além de utilizar tabelas (que não se mostram adequadas) para representar modelos de mineração de dados.

Na área de tratamento de metadados entre ambientes de manipulação e processamento de dados e conhecimento, foi adotado pelo Object Management Group (OMG) o Common Warehouse Metamodel (CWM) [POO2002], um recente padrão para intercâmbio de metadados nos ambientes de data warehousing e análise de negócios, provendo uma linguagem comum para descrever metadados e uma facilidade para intercâmbio de dados. Uma vez que processos de KDD geralmente são efetuados sobre diferentes fontes de dados, CWM fornece recursos para acesso e integração mais transparentes sobre estas diferentes fontes. Infelizmente este padrão, por ser ainda muito novo, mostra-se imaturo, não utilizável do ponto de vista prático, carecendo de documentação e ferramentas necessárias à sua aplicação.

Trabalho da Dissertação: Especificação de SKDQL, uma Linguagem de KDD

Esta dissertação se insere no projeto MATRIKS (Multidimensional Analysis and Textual Summarizing for Insight Knowledre Search) [FAV2000, FID2000, LIN2000], que tem seu foco voltado ao desenvolvimento de um ambiente integrado e aberto de suporte à decisão e KDD, pretendendo melhorar o estado da arte nesta área integrando uma diversidade maior de serviços computacionais do que os sistemas existentes e obedecendo a uma arquitetura de software moderna, aberta e extensível.

No ambiente do MATRIKS, um conjunto de recursos poderá ser acessado através de uma linguagem declarativa de especificação de consultas e processos de DCBD que, de forma transparente, disponibilizá todas as ferramentas de modo integrado, fazendo uso do poder multiplataforma, distribuído e aberto desta proposta de KDSE (Knowledge Discovery Support Environment). Com base no fluxo natural de manipulação de dados e resultados, SKDQL (Structured Discovery Query Language) é a proposta de linguagem para descoberta de conhecimento com primitivas que acessam de forma transparente e integrada ferramentas e recursos disponibilizados no ambiente MATRIKS.

Assim, no contexto desta proposta, o presente trabalho apresenta como contribuição a especificação de SKDQL (Structured Discovery Query Language), com cláusulas específicas para tarefas de KDD. Visando a utilização da linguagem, sua contextualização ao MATRIKS e a validação dos conceitos levantados neste trabalho, parte da especificação de SKDQL está implementada. Para o teste da implementação ora citada, o protótipo de SKDQL é efetivamente testado sobre o banco de dados de logs de um domínio de simuladores de futebol de robôs (RoboCup), desenvolvido como estudo de caso de KDD. Este domínio contém os reais problemas enfrentados numa tarefa de descoberta de conhecimento, pois seus logs após tratados oferecem uma vasta e detalhada estatística de comportamento dos times, permitindo assim que padrões sejam obtidos a partir destes dados.

Plano da Dissertação

O próximo capítulo trata do contexto da pesquisa, ou seja, dos pontos concernentes à descoberta de conhecimento em bancos de dados, apresentando gargalos do processo, o ambiente MATRIKS e o estado da arte em linguagens de especificação de consultas e/ou processos para DCBD.

Um estudo de caso do domínio da RoboCup é apresentado no terceiro capítulo, juntamente com os detalhes de sua modelagem e implementação, enfocando as etapas, dificuldades e conclusões do trabalho desenvolvido.

O quarto capítulo trata a especificação de SKDQL, sendo seus construtores e aspectos apresentados, explicados e exemplificados com tarefas reais de KDD no domínio da Robocup.

Subseqüentemente, o quinto capítulo aborda a implementação da interface SKDQL do MATRIKS, apresentando características do parser da linguagem, da sua geração de código e apresentando os testes realizados da implementação.

Finalizando, o sexto capítulo traz para discussão a conclusão de todo o trabalho desenvolvido, enfocando a contribuição gerada nesta pesquisa, as limitações e os trabalhos futuros.

2 Contexto da Dissertação

A interatividade do processo de KDD baseia-se no típico fluxo da atividade de análise e de tarefas correlatas, onde as técnicas e ferramentas têm, além de poder suficiente para executar passos complexos de busca e avaliação, recursos que permitem ao usuário direcionar, intervir, avaliar, corrigir e reestruturar passos e etapas do processo. De acordo com a complexidade e propósitos estabelecidos para o processo como um todo, uma série de passos tendem a repetir-se, onde a manutenção de controle de arquivos, bases de dados, programas, tratamento de dados, aplicação de algoritmos etc. são alguns pontos executados de forma iterativa ao longo do desenvolvimento de todo o trabalho.

2.1 Descoberta de Conhecimento em Bancos de Dados

O processo de tratamento e análise destas bases de dados naturalmente leva a um contexto onde a busca torna-se cada vez mais sistemática, onde os novos conhecimentos indicam novos caminhos no próprio processo, levando a investigação a ser realizada sob novos e diferentes paradigmas, caracterizando assim um cenário plenamente exploratório nestas tarefas. Neste contexto, o conhecimento do domínio explorado assume papel relevante, onde detalhes garimpados podem tornar-se grandes avanços na busca mediante a experiência de um especialista no assunto, tornando as habilidades cognitivas (de todos os envolvidos no processo) um instrumento precioso na identificação de informações e do conhecimento.

2.1.1 O Processo de Descoberta de Conhecimento em Bancos de Dados

Descoberta de conhecimento em bancos de dados, é o processo não trivial de identificar em dados padrões que sejam válidos, novos (previamente desconhecidos), potencialmente úteis e compreensivos, visando melhorar o entendimento de um problema ou um procedimento de tomada de decisão [FAY1996]. Examinando estes termos individualmente:

- Dados: conjunto de fatos F, como instâncias de um banco de dados. Por exemplo, uma coleção de *n* cadastros de pessoas físicas contendo idade, profissão, renda etc.

- Padrão: expressão E em uma linguagem L descrevendo fatos em um subconjunto F_E de F. E é dito um padrão se é mais simples do que a enumeração de todos os fatos em

F_E. Por exemplo, o padrão: "Se renda < \$*r* então a pessoa não recebe financiamento" seria aplicável para uma escolha apropriada de *r*.

- Processo: geralmente em KDD, processo é uma seqüência de vários passos que envolve preparação de dados, pesquisa de padrões, avaliação de conhecimento, refinação envolvendo iteração e modificação.

- Validade: os padrões descobertos devem ser válidos em novos dados com algum grau de certeza. Uma medida de certeza é uma função *C* mapeando expressões em *L* para um espaço de medidas M_C . Por exemplo, se um limite de padrão de crédito é ampliado, então a medida de certeza diminuiria, uma vez que mais financiamentos seriam concedidos a um grupo até então restrito a esta operação.

- Novo: em geral, assume-se que "novidade" pode ser medida por uma função *N*(E,F), que pode ser uma função booleana ou uma medida que expresse grau de "novidade" ou "surpresa". Exemplo de um fato que não é novidade: sejam E = "usa tênis" e F = "alunos de colégio" então N(E,F) = 0 ou N(E,F) = *false.* Por outro lado: sejam E = "bom pagador" e F = "trabalhador da construção civil" então N(E,F) = 0,85 ou N(E,F) = *true*.

- Potencialmente útil: padrões devem potencialmente levar a alguma atitude prática, conforme medido por alguma função de utilidade. Por exemplo, regras obtidas no processo podem ser aplicadas para aumentar o retorno financeiro de uma instituição.

- Compreensível: um dos objetivos de KDD é tornar padrões compreensíveis para humanos, visando promover uma melhor compreensão dos próprios dados. Embora seja um tanto subjetivo medir compreensibilidade, um dos fatores freqüentes é a medida de simplicidade. O fator de compreensão dos dados está relacionado à intuitividade da representação destes, bem como da granularidade alta o suficiente para que estes sejam compreendidos. Por exemplo: o log de um servidor Web não é uma representação compreensível; já fatos estatísticos extraídos deste log, tais como totais de acesso ou classificação dos acessos realizados, fornecem informação num formato mais intuitivo e de granularidade humanamente compreensível.

2.1.2 Etapas do Processo de Descoberta de Conhecimento em Bancos de Dados

O processo de KDD é interativo, iterativo, cognitivo e exploratório, envolvendo vários passos (Figura 1) com muitas decisões sendo feitas pelo analista (que é um especialista do domínio dos dados, ou um especialista de análise dos dados), conforme descrito:

1. Definição do tipo de conhecimento a descobrir, o que pressupõe uma compreensão do domínio da aplicação bem como do tipo de decisão que tal conhecimento pode contribuir para melhorar.

2. Criação de um conjunto de dados alvo (Selection): selecionar um conjunto de dados, ou focar num subconjunto, onde a descoberta deve ser realizada.

3. Limpeza de dados e pré-processamento (Preprocessing): operações básicas tais como remoção de ruídos quando necessário, coleta da informação necessária para modelar ou estimar ruído, escolha de estratégias para manipular campos de dados ausentes, formatação de dados de forma a adequá-los à ferramenta de mineração.

4. Redução de dados e projeção (Transformation): localização de características úteis para representar os dados dependendo do objetivo da tarefa, visando a redução do número de variáveis e/ou instâncias a serem consideradas para o conjunto de dados.

5. Mineração de dados (Data Mining): selecionar os métodos a serem utilizados para localizar padrões nos dados, seguida da efetiva busca por padrões de interesse numa forma particular de representação ou conjunto de representações; busca pelo melhor ajuste dos parâmetros do algoritmo para a tarefa em questão.

6. Interpretação dos padrões minerados (Interpretation/Evaluation), com um possível retorno aos passos 1-6 para posterior iteração.

7. Implantação do conhecimento descoberto (Knowledge): incorporar este conhecimento à performance do sistema, ou simplesmente documentá-lo e reportá-lo às partes interessadas.

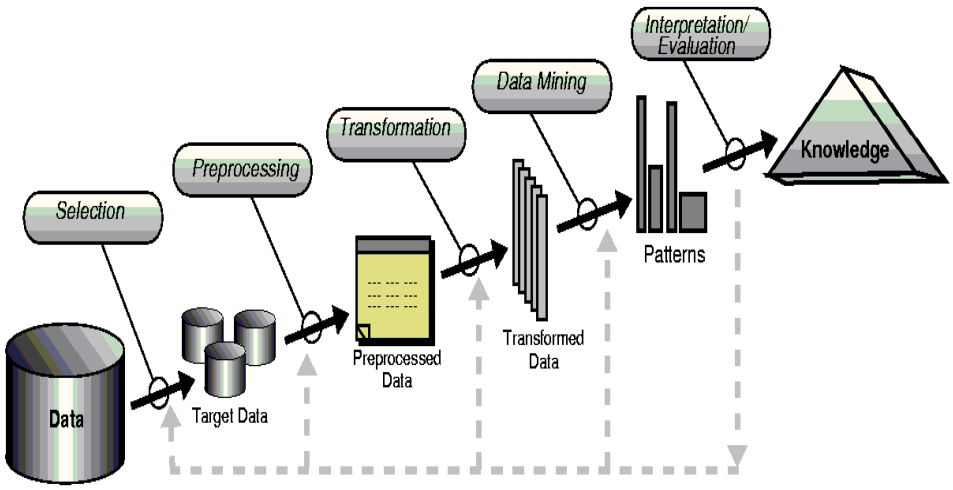

Figura 1: Etapas de KDD [FAY1996]

2.1.3 Aplicabilidade de Descoberta de Conhecimento em Bancos de Dados

Visando uma exemplificação da aplicabilidade de KDD, são apresentados a seguir casos onde a descoberta de conhecimento em bancos de dados pode desempenhar tarefas relevantes [WIT2000]:

- Submissões a empréstimos demandam do proponente o fornecimento de dados pessoais e financeiros relevantes. Estas informações são utilizadas pelas instituições financeiras como base para a decisão de efetuar ou não o empréstimo. Tal decisão é comumente tomada em dois estágios. Primeiro, métodos estatísticos são utilizados para determinar situações bem definidas em relação à aceitação ou rejeição do pedido. Os casos remanescentes, ou seja, aqueles que estão no limite necessitam de análise humana. KDD pode ser aplicado neste problema da seguinte forma: suponha-se a disponibilidade de um banco de dados histórico sobre clientes da instituição, com aproximadamente 5000 cadastros contendo 20 diferentes atributos, tais como idade, tempo de serviço, vencimentos, bens, status atual de crédito etc. O tratamento dessas informações por métodos de KDD geraria automaticamente regras objetivas e claras sobre características fundamentais a bons e maus clientes, podendo estas regras serem aplicadas para aumentar a taxa de sucesso das operações de empréstimo.

- Diagnóstico é uma das principais aplicações de sistemas especialistas. A manutenção preventiva de dispositivos eletromecânicos pode evitar falhas que interrompam processos industriais. Técnicos regularmente inspecionam cada dispositivo, medindo vibrações e outros fenômenos que indicam necessidade de manutenção. Instalações de indústrias químicas chegam a utilizar mais de mil diferentes dispositivos, que vão de pequenas bombas a grandes turbo-alternadores. A medição de vibrações e demais fenômenos é muito ruidosa, devido às limitações dos procedimentos de medição e registro. Estes dados, uma vez estudados por um especialista, conduzem a um diagnóstico. As limitações dos procedimentos técnicos, aliadas à subjetividade humana, oferecem uma margem de erro preocupante. Por outro lado, um universo de 600 falhas, cada uma devidamente registrada com seus conjuntos de medições representando 20 anos de experiência, pode ser utilizado para determinar o tipo de falha através de procedimentos de KDD, aperfeiçoando assim o processo de busca e correção de problemas eletromecânicos.

- Desde o princípio da tecnologia de satélites, cientistas ambientais tentam detectar manchas de óleo a partir de imagens de satélite, com o intuito de alertar e tomar providências rapidamente contra desastres ambientais. Estas imagens fornecem uma oportunidade para monitorar águas litorâneas dia e noite, independentemente de condições atmosféricas. Manchas de óleo aparecem como regiões escuras na imagem cujo tamanho e forma modifica-se dependendo do clima e condições marítimas. Entretanto, outras regiões negras semelhantes podem ser causadas por fatores climáticos, tais como ventos altos. Detecção de manchas de óleo é um processo manual de alto custo, que requer pessoal altamente treinado para avaliar cada região na imagem. Sistemas de detecção têm sido desenvolvidos para selecionar imagens para subseqüente processamento manual. Entretanto, é necessário ajustá-los detalhadamente para circunstâncias distintas. Descoberta de conhecimento em bancos de dados permite que estes sistemas sejam treinados para fornecer padrões de manchas e de ausência delas, permitindo ainda ao usuário controlar compromissos entre manchas não detectadas e falsos alarmes.

2.2 Gargalos no Processo de Descoberta de Conhecimento em Bancos de Dados

Descoberta de conhecimento está longe de se tornar um processo padrão com um conjunto bem definido de tarefas a serem realizadas. Além disso, sistemas computacionais que operam em domínios distintos, com bases de dados heterogêneas, distribuídas e com altas taxas de imprecisão nestes dados são potenciais alvos de projetos de descoberta de conhecimento.

Neste contexto, para sistemas de KDD, gargalos estão sempre presentes quando ocorre ausência das características que se seguem:

- Suporte para plataformas heterogêneas: possibilidade de utilizar wrappers para integrar sistemas legados, sendo estes wrappers implementados em linguagem independente de plataforma, permitindo que a ferramenta rode em plataformas diferentes.

- Eficiência e performance: a escalabilidade é um requisito fundamental, haja vista que KDD lida com as grandes massas de dados necessárias à extração de padrões que sejam ao mesmo tempo válidos, previamente desconhecidos e, na prática, úteis.

- Modularidade e integração: um sistema de KDD deve apresentar modularidade em seus componentes, visando facilitar a adição, remoção ou atualização de recursos, permitindo que o sistema seja extensível.

A abordagem destes pontos atesta a um sistema de KDD grande poder no que diz respeito à extensibilidade e interoperabilidade, abandonando os padrões dos modelos monolíticos que limitam os sistemas de KDD.

Outro ponto relevante é a funcionalidade destes sistemas no que diz respeito ao gerenciamento de conhecimento, questão abordada a seguir.

2.2.1 Gerenciamento do Conhecimento

Conhecimento, dentre outras definições, é uma crença pessoal justificada que aumenta a capacidade de um indivíduo de tomar atitudes efetivas [ALA1999]. Fatos, procedimentos, conceitos, interpretações, idéias, observações, julgamentos são exemplos de instâncias do conhecimento. Este é articulado ou comunicado a outros em forma de textos, de discurso, de

formatos computacionais ou outros meios. O receptor do conhecimento pode então cognitivamente processar e interiorizá-lo. A criação e a transferência do conhecimento tem comumente ocorrido através de várias maneiras: interações pessoais, aconselhamento, revezamento de funções, treinamentos, informações sistematizadas.

Gerenciamento do conhecimento refere-se, então, a um processo sistêmico e devidamente especificado para aquisição, organização e comunicação tácita e explícita do conhecimento de modo que outros possam fazer uso deste para atingir objetivos pré-definidos. Descoberta, captura, integração e transmissão do conhecimento configuram-se em grandes desafios deste contexto. De fato, o conhecimento é de limitada valia se não for disponível e compartilhado, enquanto que, por outro lado, a habilidade de integrar e aplicar conhecimento especializado é fundamental e extremamente valiosa. À medida que instituições tornam-se cada vez mais globais e migram para modelos de negócios cada vez mais virtuais, gerenciamento do conhecimento figura como ponto chave e estratégico das organizações.

Conseqüentemente, sistemas projetados especificamente para facilitar a codificação, coleção, integração e disseminação do conhecimento têm tido demanda crescente, devido à própria necessidade das instituições serem flexíveis e responderem com mais velocidade aos cenários dinâmicos, sendo inclusive mais inovadoras, além de aperfeiçoarem produção e tomada de decisão.

Diante deste quadro, sistemas de KDD devem ser integrados ou pelo menos facilmente interoperáveis com sistemas de gerenciamento do conhecimento. Analistas de dados e engenheiros do conhecimento demandam interfaces apropriadas para diferentes passos de tratamento do conhecimento. O novo conhecimento adquirido pode ser alvo de tarefas que envolvam representação, persistência, integração, ou transmissão deste. Observa-se, entretanto, que recursos para estas operações raramente são contemplados nas ferramentas de KDD, nem sequer a nível de interoperabilidade, uma vez que formatação, manipulação, armazenamento, reaplicação e execução de tarefas de KDD sobre o próprio conhecimento não são encaradas como funções comuns aos sistemas que se aplicam ao conhecimento, mas sim como tarefas extremamente especializadas daqueles sistemas que as implementam, subestimando o poder da integração de recursos.

2.2.2 Ambientes Integrados de KDD

Os KDSE's (Knowledge Discovery Support Environments) são sistemas integrados de descoberta de conhecimento em bancos de dados que possuem recursos e ferramentas visando o ciclo de tarefas de KDD. A comunidade científica e a indústria de software têm desenvolvido KDSE´s e metodologias com diferentes abordagens e enfoques [GOE1999, CRI2000], dos quais destacam-se alguns dos trabalhos relevantes abaixo relacionados:

- **DBMiner**

O DBMiner [DBM2000], é uma ferramenta de mineração de dados comercial que surgiu do trabalho de pesquisadores da Simon Fraser University (Canadá). A solução comporta-se como uma camada superior do Analysis Manager (ferramenta OLAP do SGBD Microsoft SQL Server), realizando suas tarefas de mineração (classificação, previsão, regressão, associação, clustering e exploração de dados) sobre cubos OLAP de um data mart ou data warehouse. Seu ambiente disponibiliza um wizard que orienta o usuário durante o processo de mineração.

A ferramenta manipula grandes bases de dados (cubos OLAP), apresentando escalabilidade, além de possuir recursos gráficos de visualização de resultados. Sua arquitetura não é aberta, impedindo a agregação de novas funcionalidades e algoritmos de mineração de dados, bem como a utilização de suas funcionalidades em software externo ao seu ambiente.

- **IBM Intelligent Miner for Data**

A ferramenta da IBM [IBM2001] desempenha suas tarefas sobre o DB2, seu gerenciador de banco de dados, tendo como enfoque o tratamento de grandes volumes de dados, haja vista sua clientela incluir empresas e instituições de grande porte com bases de dados tradicionalmente gigantescas**.**

Suas ferramentas incluem classificação, clustering, previsão, regressão, e associação. Disponibiliza recursos para pré-processamento de dados além de utilizar também redes neurais, possuindo interface ao usuário baseada em Java [COR1997]. O sistema fornece suporte à arquitetura cliente-servidor, bem como a processamento paralelo. Sua arquitetura também não é aberta, impedindo a adição de novos componentes e recursos, não sendo

possível o acesso externo a suas funcionalidades. Outra questão relevante é a ausência de uma linguagem de especificação de tarefas de KDD.

- **Darwin**

Este software da Oracle [ORA2001], que hoje trabalha sobre o seu banco de dados Oracle 9i, propõe-se a tratar grandes volumes de dados oferecendo escalabilidade e desempenho. Existe também a opção para conexões a outros SGBD´s via ODBC [MIC2001d] ou a arquivos texto. O sistema disponibiliza uma interface gráfica para o usuário, fornecendo recursos de análise e visualização de resultados (gráficos, histogramas etc.), além de possuir documentação on-line e wizards que auxiliam as tarefas de KDD. O Darwin possui suporte a arquitetura cliente-servidor, suportando também processamento paralelo em diferentes plataformas, tendo sido implementado em C++.

Dentre suas tarefas de KDD encontram-se recursos para pré-processamento de dados, previsão, regressão, classificação, clustering, associação, visualização, análise de dados exploratória, incorporando também redes neurais, além de um módulo de análise estatística. São disponibilizadas funções do sistema através de uma API Java. Não é disponibilizado no sistema nenhuma linguagem de consulta de mineração de dados para o desenvolvimento das tarefas de KDD.

- **Clementine**

O Clementine [SPS2001] é uma ferramenta da SPSS Inc. Em sua arquitetura cliente-servidor de três camadas, os clientes possuem o módulo de interface com usuário, além de um Servidor Clementine que atua como middleware acessando diretamente os servidores de banco de dados, capaz de efetuar armazenamento de resultados intermediários. A ferramenta suporta vários SGBD´s (Oracle, Informix, Sybase entre outros) estabelecendo conexão, por exemplo, via ODBC.

O software possui recursos para pré-processamento de dados (filtragem, sampling) e para tarefas de mineração: previsão, regressão, classificação, clustering, associação, visualização de modelo e análise de dados exploratória, além de uma interface interativa. Semelhantemente a outras ferramentas, Clementine não possui uma arquitetura aberta, inviabilizando extensibilidade e limitando o usuário às suas funcionalidades pré-definidas.

- **WEKA**

Waikato Environment for Knowledge Analysis – WEKA [WAI2001, WIT2000] é uma ferramenta de KDD que contempla uma série de algoritmos de preparação de dados, de aprendizagem de máquina (mineração) e de validação de resultados:

Recursos para Preparação de Dados

- Recursos de pré-processamento e transformação de dados: algoritmos de amostragem (que permitem minerar subconjuntos amostrais de grandes bases de dados); filtros (direcionados à eliminação de instâncias desnecessárias/prejudiciais à mineração); normalizadores (utilizados no encaixe de valores de atributos dentro de uma determinada faixa, por exemplo, entre 0.0 e 1.0); discretizadores (aplicáveis na redução do número de valores para um determinado atributo contínuo); conversores de dados (para migração entre formatos de dados distintos).

Recursos para Mineração de Dados

Aprendizagem Supervisionada

Esta categoria de algoritmos possui esta denominação porque a aprendizagem do modelo é supervisionada, ou seja, é fornecida uma classe à qual cada amostra no treinamento pertence. Estes algoritmos são *preditivos*, pois suas tarefas de mineração desempenham inferências nos dados com o intuito de fornecer previsões ou tendências, obtendo informações não disponíveis a partir dos dados disponíveis:

- *Classificação*: através destes algoritmos supervisionados (com ênfase na precisão da regra) é possível determinar o valor de um atributo através dos valores de um subconjunto dos demais atributos da base de dados. Por exemplo, num conjunto de dados comerciais deseja-se descobrir qual o perfil dos clientes que *consomem cosméticos importados.* Com classificadores pode-se inferir (prever) que "clientes do *sexo feminino*, com *renda superior a R$ 1.500,00* e com *idade acima de 30 anos* compram *cosméticos importados. N*este caso, o atributo *compra cosmético importado* é denominado classe, pois é o atributo alvo da classificação (cujos possíveis valores, neste caso, são "sim" ou "não"). As formas mais comuns de representação de conhecimento dos algoritmos de classificação são regras e árvores. Os algoritmos Id3,

C45, J48, ADTree, UserClassifier, PredictionNode, Splitter, ClassifierTree, M5Prime, por exemplo, geram como resultado *árvores de classificação*, enquanto que outros como Prism, Part, OneR geram *regras de classificação*. Outra opção seria a representação através de *tabela de decisão* implementada, por exemplo, pelo algoritmo DecisionTable. Modelos matemáticos, de regressão e redes neurais também representam resultados de algoritmos como SMO, LinearRegression, Neural, dentre outros.

- *Seleção de atributos*: em bases de dados encontram-se atributos que têm um peso maior ou até determinante nas tarefas de mineração de dados. Por exemplo, no caso do cliente, a sua renda com certeza é um atributo determinante nos seus hábitos de consumo. Com algoritmos de seleção de atributos é possível determinar os atributos de fato relevantes para a mineração dos dados, separando-os dos atributos irrelevantes, como por exemplo nome do cliente (que neste caso não influencia seus hábitos de consumo). O Weka disponibiliza vários algoritmos para esta categoria de mineração, dentre eles InformationGain, PrincipalComponents e ConsistencyEval.

Aprendizagem Não-Supervisionada

Nestes algoritmos o rótulo da classe de cada amostra do treinamento não é conhecida, e o número ou conjunto de classes a ser treinado pode não ser conhecido a priori, daí o fato de ser uma aprendizagem não-supervisionada. Além disso são também *descritivos*, pois descrevem de forma concisa os dados disponíveis, fornecendo características das propriedades gerais dos dados minerados:

- *Associação*: quando a classe de uma tarefa de mineração não é determinada como no caso da classificação, uma boa opção é o algoritmo de associação Apriori do Weka. Ele é capaz de gerar regras do tipo: clientes do sexo *masculino*, *casados*, *com renda superior a R$ 1.800,00* têm o seguinte *hábito de consumo*: *roupas de grife*, *perfumes nacionais*, *relógios importados.* Esta regra teria a seguinte representação: $sexo(X,[masc]) \wedge est\_civil(X,[casado]) \wedge renda(X,[1800,\infty]) \Rightarrow consome(X,[roupa\_grife, pefume\_nacional, relógio\_importado])$. Neste caso, o próprio algoritmo elege os atributos determinantes (lado esquerdo da regra) e os atributos resultantes (lado direito) na tarefa revelando associações entre valores dos atributos, tendo o algoritmo sua ênfase no compromisso entre precisão e cobertura.

- *Clustering*: em algumas situações, torna-se necessário verificar como as instâncias de uma determinada base de dados se agrupam devido a características intrínsecas de seus atributos, sem que seja definida uma classe para a tarefa. A partir da definição de uma métrica de similaridade para cada atributo e uma função de combinação destas métricas em uma métrica global, os objetos são agrupados com base no princípio da maximização da similaridade intraclasse e da minimização da similaridade interclasse. Weka possui os algoritmos Cobweb, Simple Kmeans e Em para tarefas que demandam a descoberta de padrões de agrupamento nos dados. Como exemplo, podemos utilizar algoritmos de *clustering* para identificar subgrupos homogêneos de clientes de uma determinada loja.

Recursos para Validação de Resultados

- O sistema possui recursos e funcionalidades para avaliar e comparar resultados e modelos, dentre os quais: teste e validação, que fornecem parâmetros de validade e confiabilidade nos modelos gerados (cross validation, suplied test set, use training set, percentage split); indicadores estatísticos para auxiliar a análise dos resultados (matriz de confusão, índice de correção e incorreção de instâncias mineradas, estatística kappa, erro médio absoluto, erro relativo médio, precisão, F-measure, dentre outros).

WEKA foi desenvolvido na Universidade de Waikato na Nova Zelândia, sendo escrito em Java e possuindo código aberto disponível na Web. A equipe tem lançado periodicamente correções e releases do software, além de manter uma lista de discussões acerca da ferramenta. Grande parte de seus componentes de software são resultantes de teses e dissertações de grupos de pesquisa desta universidade. Inicialmente, o desenvolvimento do software visava a investigação de técnicas de aprendizagem de máquina, enquanto sua aplicação inicial foi direcionada para a agricultura, uma área chave na economia da Nova Zelândia.

O sistema possui uma interface gráfica amigável e seus algoritmos fornecem relatórios com dados analíticos e estatísticos do domínio minerado. Grande parte de seus recursos é acessível via sua GUI, sendo que os demais podem ser utilizados programaticamente através de API´s. Foi disponibilizada também uma abrangente documentação online do código fonte. A ferramenta não possui capacidade de tratar grandes volumes de dados, comprometendo sua escalabilidade, além de não possuir uma linguagem de especificação de consultas e processos

em KDD. Via JDBC, foi possível minerar diretamente as tabelas relacionais do SQL Server do banco de dados RoboCup utlilizando o Weka.

- **Microsoft SQL Server**

O SQL Server [MIC2002] é o Sistema Gerenciador de Banco de Dados Relacional produzido pela Microsoft que incorpora diferentes recursos, dentre eles:

- Serviços de análise de dados com ferramentas integradas para tratamento, modelagem e construção de cubos de dados, bem como recursos para tarefas OLAP e linguagem (MDX) para consulta a estes cubos;

- OLE DB for Data Mining: extensão de OLE DB que suporta operações de mineração de dados (criação de um data mining model object, inserção de dados de treinamento no modelo e treinamento do mesmo, e a utilização efetiva do data mining model) sobre bancos de dados relacionais e cubos de dados em provedores OLE DB (mencionada adiante na seção Estado da Arte). Esta extensão possui apenas dois métodos de mineração embutidos: o algoritmo de classificação árvore de decisão e o algoritmo de clustering;

- Serviços de transformação de dados de diferentes fontes e formatos, permitindo extrair, transformar e carregar dados heterogêneos usando OLE DB, ODBC ou arquivos texto em qualquer banco de dados ou formato multidimensional suportado por OLE DB; permite ainda a elaboração e agendamento de scripts para automatização de tarefas;

- Suporte a padrões Web: armazenamento e recuperação de dados em formato XML, acesso a dados via Web com envio de queries via HTTP, manipulação de cubos de dados via navegador;

- Suporte a distribuição de bancos de dados em diferentes servidores e a failover (no caso de uma falha de dispositivos, os dados são imediatamente desviados para um servidor alternativo, sem que ocorra a interrupção de processos);

As tabelas geradas pelo gerador de dados (abaixo descrito) foram armazenadas no SQL Server, sendo assim toda a base de dados gerenciada pelo SGBD. A ferramenta de análise de dados mais utilizada foi o Analysis Manager, presente nas fases de modelagem,

carga e consulta aos cubos de dados (carregados diretamente a partir das tabelas relacionais). A linguagem SQL do SGBD também foi utilizada para a avaliação e comparação de resultados obtidos na fase de mineração dos dados. Sobre os cubos (Primito e Derivado), operações OLAP e demais tarefas de KDD foram aplicadas, inclusive através da utilização das demais ferramentas de análise e mineração de dados.

Devido ao tamanho da base de dados, o SGBD apresentou limitações e problemas durante o processo, dentre as quais destacam-se: baixo desempenho no processamento de grandes tabelas, incapacidade de executar consultas SQL de maior complexidade, documentação insuficiente e dificuldades na utilização de OLE DB for Data Mining.

2.2.3 Conclusão

Diante do exposto, verifica-se que uma característica desejável dos sistemas de KDD é o suporte interativo e ad hoc das tarefas de mineração de dados, proporcionando flexibilidade e eficiência na descoberta de conhecimento, através de um ambiente aberto e integrado. Uma linguagem declarativa e intuitiva de consulta [MEN2000], através de uma arquitetura aberta e facilmente extensível vem ao encontro desta questão, que é a proposta de SKDQL. Como exemplo deste fato, verifica-se que os SGBD's relacionais que têm dominado o mercado do processamento de transações por décadas devem em grande parte seu sucesso à padronização das linguagens de consulta relacionais [HAN2001].

Nesta direção, a evolução dos sistemas de KDD para processamento analítico deveria seguir passos semelhantes à evolução dos SGBD's, possuindo uma linguagem de consulta declarativa e intuitiva, baseada numa arquitetura aberta para incorporação de aspectos relevantes ao processamento, possibilitando a cobertura de todos os passos do processo de descoberta de conhecimento em bancos de dados. Neste contexto, a proposta de DMQL (Data Mining Query Language - abordada nos próximos capítulos) apresenta-se como primeiro passo interessante, porém incompleto.

Com relação à distribuição e heterogeneidade das fontes de dados das tarefas de KDD, CWM (Common Warehouse Metamodel) [POO2002] figura como proposta excelente para solução destes problemas, porém muito recente para ser de fato utilizada no contexto apresentado.

Diante do conjunto de fatores e ferramentas de KDD apresentados, além dos ambientes de suporte e propostas para KDD abordados, cabe neste ponto uma análise comparativa dos diferentes softwares voltados à descoberta de conhecimento, no que diz respeito à variedade de dados de entrada, de algoritmos de mineração implementados, de plataformas suportadas, escalabilidade, integração de ferramentas, interfaces programáticas, e de linguagem de especificação de consultas e processos (Figura 2).

	SQL Server	Weka	DBMiner	Darwin	Intelligent Miner
Variedade de Dados de Entrada	+ +	+/-	-	+	+
Variedade de Modelos Minerados	-	+ +	+	+/-	+/-
Variedade de Algoritmos de Mineração	-	+ +	-	-	-
Variedade de Plataformas	-	+ +	-	+ +	+
Escalabilidade	+	+/-	+	+ +	+ +
Integração de Ferramentas	+/-	+	+	+	+
GUI	+	+	+ +	+	+
Interface Programática	+/-	+ +	-	+ +	-
Linguagem de Especificação de Consultas e Processos	+/-	-	+/-	-	-
Gerenciamento do Conhecimento	-	-	-	-	-

Figura 2: Ambientes de suporte a DCBD – quadro comparativo

2.3 Contexto da dissertação: O Ambiente MATRIKS de Descoberta de Conhecimento em Bancos de Dados

2.3.1 Princípios de Projeto

O projeto MATRIKS (Multidimensional Analysis and Textual Summarizing for Insight Knowledre Search) [FAV2000, FID2000, LIN2000] – sendo atualmente desenvolvido no Centro de Informática da UFPE sob a coordenação do Professor Dr. Jacques Robin - visa a criação de um ambiente integrado, abrangente e aberto de suporte a decisão e KDD (Knowledge Discovery in Databases). Este projeto tem como objetivo atender a diferentes carências dos ambientes de KDD, relacionadas à integração de ferramentas no processo, ao gerenciamento do conhecimento do modelo minerado, à linguagem de especificação de consultas/processos e à variedade de dados de entrada, modelos e algoritmos de mineração .

O MATRIKS pretende melhorar o estado da arte em sistemas de suporte a decisão e KDD de duas maneiras: (1) integrando, em um único ambiente, uma diversidade maior de serviços computacionais do que os sistemas existentes e (2) ser desenvolvido segundo uma arquitetura de software moderna, aberta e extensível por meio de encapsulamento de serviços em componentes, comunicando-se através de interface de software aplicativo (ou API do inglês, Application Program Interface). A arquitetura dos sistemas atuais de suporte a decisão e KDD é monolítica, o que impede tanto sua extensão com novos serviços pelo usuário como sua comunicação com software externo em um sistema maior. A falta de integração dos vários serviços computacionais necessários ao processo exploratório, iterativo e interativo de KDD constitui hoje o maior gargalo nesse contexto.

As tecnologias contempladas para integração no projeto MATRIKS são data warehousing, mineração de dados, bancos de dados dedutivos orientados a objetos e geração automática de hipertextos em linguagem natural.

2.3.2 OLAP e Data Warehousing

"Um Data Warehouse é uma coleção de dados orientada por assuntos, integrada, variante no tempo e não volátil, no suporte aos processos de tomada de decisão de gerência" [INM1996, KIM1996] . A saber:

- Orientada por assuntos: significa que o Data Warehouse (DW) armazena informações sobre conceitos específicos (o cliente, as vendas etc.) importantes para o negócio da empresa e não tarefas de processamento.

- Integrada: significa que os dados são armazenados em um formato consistente, onde conflitos de armazenamento e de modelagem devem ser resolvidos uma vez que, potencialmente, dados de diferentes fontes e diferentes modelos compõem o data warehouse. Técnicas de limpeza e integração de dados são utilizadas para garantir a consistência destes.

- Variante no tempo: significa que os dados são temporais, ou seja, dados que armazenam o histórico das modificações de valores ao longo do tempo ao invés de armazenarem apenas o valor mais atual.

- Não volátil: significa que os dados não mudam uma vez armazenados no DW, sendo acessados apenas para leitura. Duas operações são realizadas no DW: carga de dados e acesso aos dados.

A maioria das tarefas dos bancos de dados operacionais é desempenhar transações on-line e processamento de queries. Estes sistemas são denominados de OLTP (on-line transaction processing). Por outro lado, OLAP (on-line analytical processing) são sistemas (ferramentas) que organizam e apresentam os dados do data warehouse em vários formatos com o intuito de atender às diferentes necessidades dos usuários.

2.3.3 Programação em Lógica e Bancos de Dados Dedutivos

Bancos de dados e programação em lógica são duas áreas desenvolvidas de forma independente na ciência da computação. A tecnologia de banco de dados tem se desenvolvido para organizar, gerenciar e manter eficientemente e efetivamente grandes volumes de dados complexos em crescimento de forma confiável em vários dispositivos de memória. A base estrutural de bancos de dados tem sido o foco primário para pesquisa que tem levado ao desenvolvimento de vários modelos de dados. O modelo de dados mais conhecido e utilizado é o relacional. O poder do modelo de dados relacional baseia-se em seus fundamentos matemáticos rigorosos com um paradigma simples a nível de usuário e linguagens de queries de alto-nível orientada a conjuntos. Entretanto, o modelo de dados relacional tem sido considerado inexpressivo para muitas aplicações novas de bancos de dados.

Programação em lógica é um crescimento direto dos trabalhos anteriores em prova automática de teoremas e inteligência artificial. É baseada em matemática lógica, que é o estudo das relações entre asserções e deduções, sendo formalizada em termos de prova e teorias de modelos [LIU1999]. A programação em lógica utiliza lógica para representar conhecimento e dedução para resolver problemas através da derivação de conseqüências lógicas.

Estudos importantes nas relações entre programação em lógica e bancos de dados relacionais têm sido conduzidos, e estas duas áreas são consideradas semelhantes no que diz respeito à representação de dados no nível da linguagem. Elas também são consideradas complementares em muitos aspectos. Sistemas de bancos de dados relacionais são superiores aos padrões de implementação de Prolog com respeito à independência de dados, acesso a

armazenamento secundário, concorrência, recuperação, segurança e integridade. Entretanto, o poder expressivo e a funcionalidade das linguagens de consulta relacionais são limitadas se comparadas às linguagens de programação em lógica. Linguagens relacionais não têm capacidade de raciocínio embutido. Prolog, por outro lado, pode ser usada com uma linguagem de programação de propósito geral. Ela pode ser usada para expressar fatos, informação dedutiva, recursão, consultas, atualizações e restrições de integridade de maneira uniforme.

A integração de programação lógica e técnicas de bancos de dados relacionais tem levado à ativa área de pesquisa dos bancos de dados dedutivos. Estes combinam os benefícios das duas abordagens, tais como uniformidade representacional e operacional, capacidades de raciocínio, recursão, consultas declarativas, acesso eficiente a armazenamento secundário, dentre outras funcionalidades.

No ambiente Matriks é utilizado o XSB [XSB2001], uma implementação do Prolog, para atender à demanda de gerenciamento do conhecimento em ambientes de KDD. Esta implementação tem sido utilizada em diferentes aplicações: especificação declarativa para processar dados na Web (Web Semântica [BER1999]), especificação e verificação de sistemas, integração de informações, dentre outras.

2.3.4 Bancos de Dados Dedutivos Orientados a Objetos

O modelo de dados orientado a objetos vem atender demandas das novas aplicações em bancos de dados, implementando conceitos de classe, herança, encapsulamento, identidade e persistência de objetos. Bancos de dados orientados a objetos (BDOO) são uma adaptação a sistemas de banco de dados do paradigma de programação orientado a objetos, sendo baseados no conceito de encapsular os dados em um objeto e o código que opera naqueles dados. Objetos estruturados são agrupados em classes, e estas por sua vez estruturadas em sub e superclasses (hierarquia de classes) [SIL1999].

Nesta ótica, verifica-se que bancos de dados orientados a objetos fornecem melhores recursos para organizar e manipular objetos estruturados, integrando componentes estruturais e comportamentais num *framework* uniforme. Entretanto, BDOO's não possuem fundamentos para lógica e consulta declarativa. Bancos de dados dedutivos oferecem inferência baseada em lógica e consultas declarativas com firmes fundamentos lógicos. Entretanto, estes carecem do poder de modelagem de dados oferecido pelos BDOO's. Bancos de dados dedutivos

orientados a objetos (BDDOO) são a proposta natural para superar estas carências mútuas [LIU1999a].

BDDOO's integram paradigmas de orientação a objetos e da lógica, atendendo a requisitos de aplicações que necessitem representar dados complexos e informação parcial (como predicados em Prolog), fornecendo raciocínio embutido, expressão de fatos, informação dedutiva, representação do conhecimento, recursão, consultas, atualizações e restrições de integridade, com uniformidade representacional e operacional.

Para atender à necessidade de gerenciamento do conhecimento no formalismo de orientação a objetos no Matriks está sendo utilizada Flora [LUD 1999], uma plataforma de linguagem orientada a objetos baseada em conhecimento para desenvolvimento de aplicações, a qual é utilizada em diferentes domínios, dentre eles: implementação de agentes inteligentes, gerenciamento de ontologias, Web Semântica.

2.3.5 Pontos de Pesquisa do MATRIKS

As principais questões abertas de pesquisa desse projeto são:

1. Integração do modelo de dados multidimensional dos servidores OLAP [PET1999] com o modelo de dados dedutivo orientado a objetos. O primeiro permite fazer consultas analíticas sobre um data warehouse oriundo de fontes de dados primitivas, como bancos de dados transacionais e arquivos de logs de servidores. O segundo possibilita operações dedutivas sobre os dados explorando as propriedades do modelo orientado a objetos (hierarquia, encapsulamento, entre outros). Essa integração inclui quatro subtarefas, para as quais o modelo dedutivo orientado a objetos é especialmente adequado: (1) derivação de um esquema de dados multidimensional analítico a partir de um esquema de dados relacional transacional, (2) integração de vários esquemas fontes, (3) carga e atualização periódica do data warehouse, a partir das suas várias fontes de dados primitivos, e (4) derivação, a partir desses dados primitivos, de agregações complexas, dependentes da aplicação e de granularidade suficientemente abstrata para a tomada de decisões. A integração dos modelos de dados multidimensional e dedutivo orientado a objetos também permite a automatização de uma quinta tarefa no processo de KDD: a exploração do data warehouse em busca de insights decisórios por meio de regras heurísticas contendo operações OLAP nas suas premissas e conclusões.

2. A integração de OLAP com a mineração de dados, resultando no OLAM (On-Line Analytical Mining). Essa integração envolve (1) a definição de várias API's permitindo a chamada de consultas OLAP em componentes de mineração de dados e vice-versa, e (2) a definição de linguagens declarativas de consulta OLAM, permitindo especificar a aplicação de diversos algoritmos de mineração de dados sobre resultados de diversas consultas OLAP. No contexto do MATRIKS, será interessante definir tal linguagem de consulta OLAM tanto no paradigma de linguagens de consultas em bancos de dados relacionais e multidimensionais como no paradigma de linguagens dedutivas orientadas a objetos.

3. A definição de regras heurísticas para exploração de data warehouses usando consultas OLAM como operações primitivas. Essas regras podem ser tanto estratégicas e independentes do domínio de aplicação, quanto táticas e específicas para um domínio particular.

4. Desenvolvimento de técnicas extensíveis e portáteis de geração automática de resumos em linguagem natural, formatada em hipertexto dos insights decisórios resultantes da exploração heurística da estrutura multidimensional e multigranularidade do data warehouse por meio de consultas OLAM. Nenhum gerador de texto em linguagem natural anterior ao projeto MATRIKS era capaz de gerar resumos em formato hipertexto. Alguns geram resumos textuais lineares e outros hipertextos sem preocupação de concisão. Além do mais, a extensibilidade e portabilidade de todos era extremamente limitada.

2.3.6 Arquitetura

A arquitetura do MATRIKS obedece aos princípios de distribuição, sendo um ambiente multiplataforma baseado em Java RMI [SUN2001a] como *middleware* de base suportando comunicação entre qualquer par de componentes (Figura 3). Como plataforma embutida, outros *middlewares* podem ser integrados, o que permite a comunicação direta entre dois componentes via Java RMI.

Além de Java RMI, ainda existem possíveis opções por: DCOM (Distributed Component Object Model) [MIC2002b], que é a plataforma Microsoft para a distribuição de componentes, CORBA [OMG2001] padrão OMG para distribuição, além da opção por um repositório compartilhado XML. A escolha por Java RMI baseia-se nos próprios recursos de distribuição disponibilizados, na sua natureza multiplataforma e orientada a objetos (para a

construção e integração de componentes de software), bem como no amplo leque de ferramentas disponíveis.

A arquitetura do MATRIKS é composta por quatro camadas:

- **Camada de Gerenciamento de Dados** (*Data Management Layer*): conforme o próprio nome, é a camada responsável por todas as tarefas de recuperação e armazenamento de dados. Inclui um SGBD Relacional (SQL Server) gerenciando sua base de dados, servidor OLAP (MS OLAP) para manipulação dos cubos OLAP e Banco de Dados Dedutivo Orientado a Objetos (Flora).

- **Camada de Middleware** (*Middleware Layer*): composta por elementos que viabilizam a comunicação entre componentes:

    - JODI [FID2000]: API Java para Interoperabilidade entre Sistemas OLAP e Servidores OLE DB for OLAP.

    - JDBC, Java Database Connectivity [SUN2002]: permite a programas Java a manipulação de instâncias de bancos de dados.

    - JB2P, API Prolog Java [ROC2001]: responsável por fazer com que Java acesse o motor de inferência Prolog, repassando-lhe tarefas de derivação e obtendo o resultado.

    - XSB Prolog [XSB2001]: banco de dados dedutivo para inferência e gerenciamento do conhecimento.

    - SKDQL: implementação em Java da especificação da linguagem.

    - Java RMI (Remote Method Invocation) [SUN2001a]: suporta a distribuição das fontes de dados.

- **Camada de Processamento** (*Processing Layer*): contém os componentes responsáveis pelo processamento propriamente dito das tarefas definidas. Formado pelo Weka [WAI2001, WIT2000], com minerador de associações, de clusters, de árvores de decisão, de regras de decisão, classificação e relacionais; inclui também OCCOM (OLAP Cube

Cell Outlier Miner) [SAR1998], componente para mineração de exceções em cubos OLAP.

- **Camada de Interface** (*Interface Layer*): possui HYSSOP (Hypertext Summary System for Olap) [FAV2000], um gerador de resumos de insights decisórios obtidos por KDD no formato hipertexto em linguagem natural; e também GUI Weka para visualização de clusters e de árvores de decisão, incluindo ainda a interface ao usuário integrada do MATRIKS.

As bases de dados para as tarefas de KDD podem ser: banco de dados transacional, flat file, log file. Os componentes das camadas de processamento e de interface comunicam-se com a camada de dados através das API´s disponibilizadas pela camada de *middleware*. Sistemas externos podem acessar as funcionalidades do sistema via a API Integrada do MATRIKS.

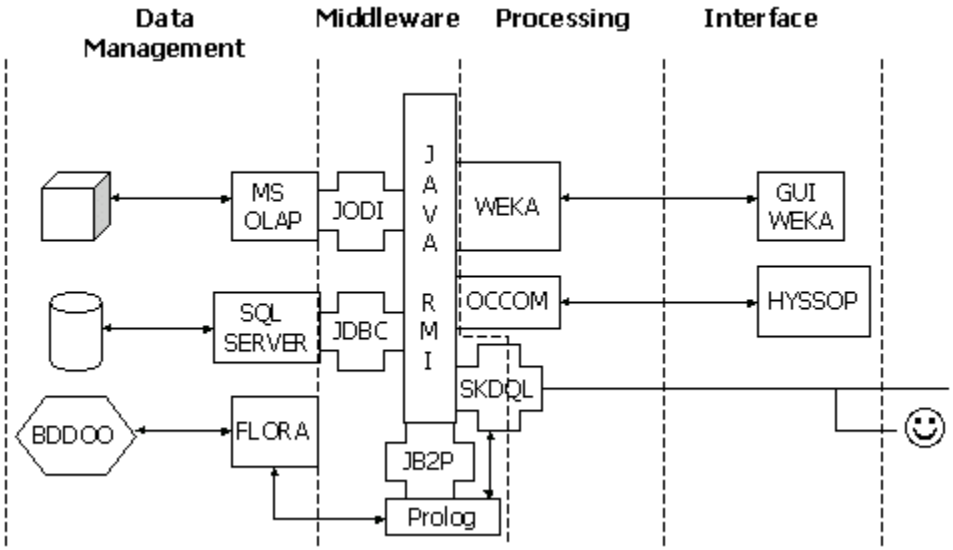

Figura 3: Sofwares envolvidos no Projeto MATRIKS

2.3.7 Estado Atual da Implementação

No seu estado atual, o projeto MATRIKS já gerou avanços importantes sobre as questões 1 e 4 da seção 2.3.5 . Para a questão 1, a infra-estrutura de software e uma especificação parcial de linguagem integrada foram desenvolvidos no escopo de duas dissertações de mestrado. Essas duas testes resultaram em JODI/DOODI (Java Olap Data

Interface / Deductive Object-Oriented Olap Data Interface) [FID2000, LIN2000] um conjunto de API entre OLE BD for OLAP (um padrão para consultas analíticas em data warehouse) e as linguagens Java, Prolog [BRA1999] e Flora [LUD1999] (uma linguagem de consulta e manipulação para bancos de dados dedutivos orientados a objetos [LIU1999]).

Para a questão 4, técnicas foram desenvolvidas e implementadas em HYSSOP (Hypertext Summary System for Olap) [FAV2000], um protótipo de gerador de resumos de insights decisórios obtidos por KDD no formato de um hipertexto em linguagem natural, resultando em uma tese de doutorado.

2.3.8 Papel de SKDQL no MATRIKS

Sendo o MATRIKS um framework para descoberta de conhecimento em bancos de dados baseado em componentes e *middleware*, dentro de uma arquitetura aberta e extensível, suportando a heterogeneidade, iteratividade e interatividade dos processos de KDD, SKDQL (Structured Knowledge Discovery Query Language) é a proposta de uma linguagem de consulta estruturada para KDD para tal *framework* ou software externo, seguindo os padrões de SQL, oferecendo primitivas para acesso, pré-processamento, mineração de dados e manipulação do conhecimento.

No MATRIKS, SKDQL encontra-se na camada de middleware acessando (via camada de gerenciamento de dados) as fontes de dados e, por outro lado, via (camada de processamento) os componentes de tratamento, mineração e visualização de informação e conhecimento. O problema do pré-processamento, que envolve passos de seleção, limpeza e transformação dos dados também é contemplado na especificação de SKDQL, permitindo que estas tarefas sejam acessadas e gerenciados pela linguagem no ambiente do MATRIKS.

Numa típica tarefa de KDD, com ferramentas não integradas, o usuário teria de interagir diretamente com diferentes ferramentas (e com seus típicos problemas de integração, configuração etc): SGBD para acesso aos dados, software para pré-processamento dos dados, mineração, dedução, visualização de informações, entre outros. No ambiente do MATRIKS, via SKDQL, o analista especificará as tarefas a serem efetuadas utilizando diretamente os recursos da linguagem, abstraindo-o de questões relacionadas às ferramentas em si, contando ainda com as vantagens da arquitetura do MATRIKS (suporte a heterogeneidade, distribuição, extensibilidade).

2.4 Estado da Arte em Linguagens de Especificação de Consultas e/ou Processos para DCBD

DMQL, OLE DB for DM e CWM são algumas das iniciativas relevantes em direção à padronização e/ou interoperabilidade dos processos de KDD.

2.4.1 DMQL – Data Mining Query Language

Uma das características desejáveis a um sistema de KDD é a habilidade deste de suportar mineração de dados ad hoc e interativa, objetivando facilitar e flexibilizar uma efetiva descoberta de conhecimento. Linguagens de consulta para mineração de dados podem suportar tais características.

A equipe de Ciência da Computação da Simon Fraser University (Canadá) desenvolveu DMQL (Data Mining Query Language) [HAN1996], que contempla primitivas para:

- conjunto de dados relevantes a ser minerado
- o tipo de conhecimento a ser minerado
- o conhecimento prévio a ser usado no processo de descoberta
- as medidas de interesse e limites para avaliação de padrões
- a representação da visualização dos padrões descobertos

DMQL (Figura 4) especifica a mineração ad hoc de vários tipos de conhecimento a partir de bancos de dados transacionais e data warehouses em vários níveis de abstração. A linguagem adota uma sintaxe semelhante a SQL, o que facilita a sua aprendizagem por parte daqueles que já conhecem SQL.

A linguagem não possui recursos para pré-processamento de dados, contando com um número limitado e fixo de algoritmos de mineração de dados. Atualmente a mesma não se encontra implementada, uma vez que sua única aplicação prática encontra-se no DBMiner, sendo utilizada como recurso de descrição de tarefas, ou seja, via um wizard o usuário define uma tarefa de mineração e, no último passo deste assistente, a ferramenta apenas apresenta cláusulas DMQL da tarefa a ser efetuada, não permitindo assim a construção de tarefas utilizando diretamente as primitivas da linguagem.

```
<DMQL> ::= <DMQL_Statement> {<DMQL_Statement>}
<DMQL_Statement> ::= <Data_Mining_Statement>
                   |    <Concept_Hierarchy_Definition_Statement>
                   |    <Visualization_and_Presentation>
<Data_Mining_Statement> ::= use database <database_name> | use data warehouse <data_warehouse_name>
                   {use hierarchy <hierarchy_name> for <attribute_or_dimension>}
                   {Mine_Knowledge_Specification)
                   in relevance to <attribute_or_dimension_list>
                   from <relation(s)/cube(s)>
                   [where <condition>]
                   [order by <order_list>]
                   [group by <grouping_list>]
                   [having <condition>]
                   {with [<interest_measure_name>] threshold = <threshold_value>
                   [for <attribute(s)>]}
<Mine_Knowledge_Specification> ::= <Mine_Char> | <Mine_Discr> | <Mine_Assoc> | <Mine_Class>
<Mine_Char> ::= mine characteristics [as <pattern_name>]
                analyse <measure(s)>
<Mine_Discr> ::= mine comparison [as <pattern_name>]
                 for <target_class> where <target_condition>
                 {versus <contrast_class_I> where <contrast_condition_i>}
                 analyze <measure(s)>
<Mine_Assoc> ::= mine associations [as <pattern_name>]
                 [matching <metapattern>]
<Mine_Class> ::= mine classification [as <pattern_name>]
                 analyze <classifying_attribute_or_dimension>
<Concept_Hierarchy_Definition_Statement> ::= define hierarchy <hierarchy_name>
                                             [for <attribute_or_dimension>]
                                             on <relation_or_cube_or_hierarchy>
                                             as <hierarchy_description>
                                             [where <condition>]
<Visualization_and_Presentation> ::= display as <result_form> | {<Multilevel_Manipulation>}
<Multilevel_Manipulation> ::= roll up on <attribute_or_dimension>
                            | drill down on <attribute_or_dimension>
                            | add <attribute_or_dimension>
                            | drop <attribute_or_dimension>
```

Figura 4: Resumo da sintaxe em alto nível de DMQL [HAN1996]

Um exemplo de consulta em DMQL seria:

```
use database Principal
in relevance to Cidade, Produto, Ano, Quantidade
from Vendas
where ano>=1990 and ano<=2000
```

```
mine characteristics as Analise
analyze Quantidade
display as table
```

Neste caso, a tabela Vendas do banco de dados Principal teria os atributos Cidade, Produto, Ano e Quantidade (de 1990 a 2000) minerados por um algoritmo de caracterização de dados, cujo atributo Quantidade seria o elemento de referência desta análise, sendo o resultado expresso em forma tabular.

2.4.2 OLE DB for Data Mining

OLE DB [MIC2001c] é uma interface de baixo nível da Microsoft que implementa uma especificação de acesso a dados via ODBC (Open Database Connectivity) [MIC2001d]. Sendo uma API para acesso universal de dados, permite que dados armazenados em diferentes repositórios (SGBD's, arquivos texto, planilhas eletrônicas, páginas Web, entre outros) possam ser manipulados. Ferramentas OLAP (On-Line Analytical Processing) são aquelas que organizam e apresentam os dados do data warehouse em vários formatos com o intuito de atender às diferentes necessidades dos usuários [KIM1996]. COM (Component Object Model) [MIC2001b] é uma arquitetura de software da Microsoft que permite que aplicações sejam construídas a partir de componentes de software.

OLE DB for OLAP [MIC2001] estende OLE DB para dados multidimensionais armazenados em data warehouses e data marts. Desta forma, aplicações OLAP podem acessar dados armazenados em fontes diversas de informação, independentemente de localização ou tipo. Adicionalmente, backends - tais como servidores OLAP e bases de dados - podem oferecer acessibilidade genérica para várias aplicações. OLE DB for OLAP é uma especificação suportada pela indústria de processamento de dados multidimensionais, representando um padrão de desenvolvimento para acesso a dados OLAP.

OLE DB for Data Mining (OLE DB for DM) [MIC2000] é uma extensão de OLE DB que suporta operações de mineração de dados sobre provedores OLE DB. A especificação é direcionada à padronização das primitivas de linguagem de mineração de dados, visando tornar-se um padrão de indústria. Ela é desenhada para permitir aplicações de mineração de dados do cliente, consumindo serviços de uma variedade de provedores ou pacotes de software de DM. OLE DB for DM descreve uma abstração do processo de mineração de dados. Como extensão de OLE DB, introduz um novo objeto virtual chamado Data Mining

Model (DMM), bem como comandos para manipulação deste objeto virtual. DMM é como uma tabela relacional, exceto pelo fato de conter colunas especiais que podem ser utilizadas para treinamento e descoberta de padrões possibilitando, por exemplo, a criação de um modelo de previsão e gerando as previsões.

Ao passo que uma tabela relacional armazena dados, DMM armazena os padrões descobertos pelo algoritmo de mineração. A manipulação de um DMM é semelhante a de uma tabela SQL. Entretanto, esta abordagem mostra-se inadequada, pois tabelas não são suficientemente flexíveis para representar modelos de mineração de dados. Além disso, o ambiente não provê mecanismos para disponibilização de funções do software a partir de um sistema externo.

As três operações principais desempenhadas pelos clientes num DMM são: criação de um data mining model object; inserção de dados de treinamento no modelo e treinamento do mesmo; e a utilização efetiva do data mining model. Estas operações, exemplificadas na especificação do OLE DB for DM [MIC2000], são apresentadas abaixo:

a) Criação de um *data mining model object*. Utilizando um comando OLE DB, o cliente executa um CREATE semelhante ao CREATE TABLE do SQL:

```
CREATE MINING MODEL [Age Prediction]
(
    [Customer ID]                       LONG    KEY,
    [Gender]                            TEXT    DISCRETE,
    [Age]                               DOUBLE  DISCRETIZED() PREDICT,
    [Product Purchases]                 TABLE
    (
           [Product Name]               TEXT    KEY,
           [Quantity]                   DOUBLE  NORMAL CONTINUOUS,
           [Product Type]               TEXT DISCRETE RELATED TO [Product Name]
    )
 )
 USING [Decision Trees]
```

b) Inserção de dados de treinamento no modelo. De forma similar ao preenchimento de uma tabela em SQL, o cliente utiliza um formato do comando INSERT INTO. O comando SHAPE é utilizado para criar a tabela aninhada:

```
INSERT INTO [Age Prediction]
(
    [Customer ID], [Gender], [Age],
    [Product Purchases](SKIP, [Product Name], [Quantity], [Product Type])
 )
SHAPE
{
    SELECT [Customer ID], [Gender], [Age] FROM Customers ORDER BY [Customer ID]
```

```
 }
APPEND
 (
    {SELECT [CustID], [Product Name], [Quantity], [Product Type]
     FROM Sales
     ORDER BY [CustID]}
    RELATE [Customer ID] To [CustID]
  )
AS [Product Purchases]
```

c) Utilização do *data mining model* para fazer algumas previsões. Previsões são realizadas com um comando SELECT que realiza um *join* de todos os conjuntos de possíveis casos do modelo com outro conjunto de casos existentes. Estes últimos podem ser incompletos. Neste exemplo, o valor de "Age" não é conhecido. Juntando estes casos incompletos ao modelo e selecionando a coluna "Age" do modelo, o retorno será a previsão de "Age" para cada caso:

```
SELECT t.[Customer ID], [Age Prediction].[Age]
FROM [Age Prediction]
PREDICTION JOIN
(
SHAPE
{
SELECT [Customer ID], [Gender], FROM Customers ORDER BY [Customer ID]
}
APPEND
(
{SELECT [CustID], [Product Name], [Quantity] FROM Sales ORDER BY [CustID]}
RELATE [Customer ID] To [CustID]
)
AS [Product Purchases]
) as t
ON [Age Prediction] .Gender = t.Gender and
[Age Prediction] .[Product Purchases].[Product Name] = t.[Product Purchases].
                                                      [Product Name] and
[Age Prediction] .[Product Purchases].[Quantity] = t.[Product Purchases].[Quantity]
```

2.4.3 CWM – Common Warehouse Metamodel

Common Warehouse Metamodel (CWM) [POO2002] é um recente padrão definido pelo Object Management Group (OMG) [OMG2002] para intercâmbio de metadados nos ambientes de data warehousing, descoberta de conhecimento em banco de dados e análise de negócios. CWM provê uma linguagem comum para descrever metadados (baseada num metamodelo genérico, mas semanticamente repleto, para data warehousing e análise de negócios) e facilidades para intercâmbio de dados e especificação de classes de processos de KDD. CWM reusa três outros padrões da OMG: UML [BOO1998], XML e MOF.

XML (Extensible Markup Language) [W3C2001] é uma linguagem padrão W3C [W3C2002] para definição de linguagens de markup (por exemplo, HTML). Ela descreve uma classe de objetos, documentos XML, que são escritos utilizando estas linguagens de markup. Esta linguagem foi desenvolvida tendo como alvo a Internet, sendo caracteristicamente modular (reuso e combinação de outros formatos), independente de plataforma, amplamente suportada e apropriada para estruturar dados. XMI (XML Meta Data Interchange) [OMG2002a] é uma linguagem XML para intercâmbio de modelos UML, meta dados completos ou fragmentos destes em sistemas de software, permitindo novos caminhos de integração entre aplicações e tecnologias, eliminando incompatibilidade entre ferramentas.

MOF (Meta-Object Facility) [OMG2001a] é uma facilidade para meta modelos da OMG, baseada em conceitos e construções de UML, apresentando-se como um framework extensível para definição de modelos para metadados. MOF não define sua própria notação gráfica ou linguagem de restrições, mas utiliza a notação UML e OCL (Object Constraint Language) [WAR1999] para tais propósitos. MOF faz a ligação entre meta modelos dissimilares provendo uma base comum para estes meta-modelos. Desta forma, se dois meta modelos são compatíveis com MOF, então modelos baseados nestes podem residir no mesmo repositório.

CWM utiliza os padrões acima descritos da seguinte forma:

- Utilização de UML para representação de modelos e meta modelos, bem como para descrever a semântica da análise de objetos e do projeto de modelos

- Utilização de MOF para definir e manipular meta modelos programaticamente utilizando interfaces CORBA [OMG2001], tendo em vista a infraestrutura de objetos distribuídos.

- Utilização de XMI para intercâmbio linear de metadados

Existem geradores automáticos de XMI e MOF para modelos UML e formatos XML. Alguns deles são ObjectTif [MCR2000], Jvision [MCH2001], *A Textual Notation Generator for MOF Models* [DST2001] e Moderes [RES2000].

CWM apresenta como grande vantagem o fato de apoiar-se em padrões consolidados (UML, XML e MOF), versáteis, independentes de linguagem, bem divulgados e com um

grande número de usuários. Estes padrões de especificação apresentam-se de forma complementar, provendo recursos gráficos para utilização por humanos, paradigma orientado a objetos para software e especificação textual, utilizável por software e humanos, facilitando *parsing* e uma utilização transparente da Internet e seus recursos.

A cobertura da especificação de CWM é bastante abrangente, consistindo de definições de metamodelos nos seguintes domínios: modelos de objetos, dados relacionais, dados multidimensionais, dados XML, transformações de dados, OLAP, mineração de dados, visualização da informação, processamento e operação de data warehouses, nomenclatura de negócios. Esta abrangência confere à especificação grande complexidade, demandando conhecimento e experiência em seus fundamentos para o seu efetivo uso. Não obstante, verifica-se a ausência de tutorais, documentações e estudos de casos suficientes para ampla compreensão da especificação e da maneira de aplicá-la em problemas práticos . Atualmente, Oracle, Unisys, DimensioEDI, entre outras empresas estão disponibilizando interfaces CWM para seus respectivos sistemas de software. Devido à própria complexidade da especificação e da atual indisponibilidade de implementações, a utilização prática de CWM não é possível neste momento.

2.4.4 Conclusão

DMQL, OLE DB for DM, CWM e outras propostas contemplam questões de KDD referentes à integração de diferentes modelos de dados, sua heterogeneidade, dinâmica dos dados e das tarefas, entre outros. A complexidade, a recente especificação de CWM e sua atual indisponibilidade a nível de software e documentação ainda figuram como um problema para sua aplicação em KDD.

Por outro lado, a proposta de CWM (que não fora ainda divulgada quando iniciado este trabalho) vem confirmar a importância do problema levantado por SKDQL, haja vista sua proposta de prover recursos para aperfeiçoar as tarefas de KDD em seu contexto heterogêneo, iterativo e interativo.

Neste cenário, confirma-se a característica desafiadora dos problemas de KDD, diante da diversidade e complexidade dos aspectos que abrange, dentre eles: integração de dados, modelos, conhecimento, ferramentas e software; especificação de processos iterativos e de conhecimento prévio; interface usuário/máquina.

3 Estudo de Caso de Tarefas de KDD: Mineração de Arquivos de Log de Futebol de Robôs

3.1 Introdução ao Futebol de Robôs Simulado da RoboCup

A copa mundial de futebol de robôs (RoboCup) [ROB2002] é uma iniciativa internacional visando estimular a pesquisa em inteligência artificial, robótica e sistemas multiagentes, através do fornecimento de um problema padrão onde um vasto conjunto de tecnologias pode ser integrado e examinado, bem como aplicado em projetos voltados à educação.

Este ambiente é um domínio complexo e realista. Este modela um sistema robótico hipotético, combinando características de diferentes sistemas e simulando jogadores de futebol humanos [STO2000]. Este simulador, atuando como um servidor (ambiente), provê um domínio e suporta usuários que desejam construir seus próprios agentes (clientes/jogadores). Programas cliente conectam-se ao servidor via soquetes UDP, cada um controlando um único jogador. Para cada partida, são simulados os movimentos de todos os objetos no ambiente, enquanto cada cliente age como um jogador, enviando comandos de movimentação ao servidor. Este servidor, chamado de Soccer Server, faz com que o jogador seja controlado pelo cliente através da execução dos comando de movimento, enviando também informação sensorial da perspectiva daquele jogador ao cliente.

A divisão de simuladores (Simulation League) da Robocup permite que jogadores virtuais realizem competições num ambiente dinâmico, complexo e multiagente. Esta proposta visa estimular pesquisadores a projetar "cérebros" (software) para os corpos simulados dos jogadores. O servidor de jogos (Soccer Server) da divisão de simuladores gera logs da posição de cada agente de software dos times durante os jogos realizados. Estes logs, após tratados, poderiam oferecem uma vasta estatística detalhada do comportamento dos times.

Este domínio de futebol de robôs foi escolhido como estudo de caso de KDD para esta dissertação por se tratar de um tema de amplo domínio (futebol) com suficientes fontes de

dados disponíveis para pesquisa. No contexto do MATRIKS e de SKDQL é um caso semi-real do processo de KDD para fazer surgir e ilustrar dificuldades práticas concretas que sempre estão presentes nos processos de KDD, além de guiar a concepção de ambos (SKDQL e MATRIKS), já que a razão de ser destes é precisamente amenizar essas dificuldades. Outro ponto está relacionado à avaliação da eficiência e à adequação dos protótipos para a tarefa de facilitação, automação, simplificação e agilização dos processos de KDD. Os dados selecionados e pré-processados foram minerados, tendo em vista a busca de padrões de comportamento e de jogadas dos simuladores, objetivando a descoberta de conhecimento em termos de táticas de jogo dos times participantes do campeonato da RoboCup de 1999.

Para cada partida, dois times de 11 clientes independentemente controlados são conectados ao servidor. O simulador inclui uma ferramenta de visualização (Figura 5), onde cada jogador é representado com um círculo com duas metades. O lado mais claro é a direção na qual o jogador está “olhando”. O simulador também inclui um juiz, que reforça as regras do jogo. Ele indica mudanças no modo de jogo, saída de bola, gol, impedimento ou final de jogo.

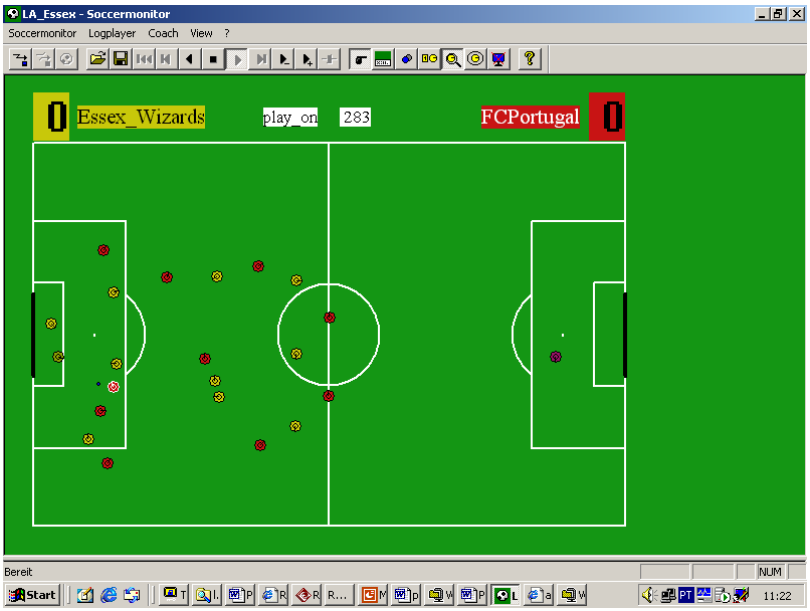

Figura 5: Soccer Monitor

Uma das complexidades do mundo real contemplada pelo soccer server é percepção e ação assíncronas. A maioria dos domínios utiliza ação e percepção síncronas: o agente percebe o mundo, age, percebe o resultado, age novamente, e assim por diante. Neste paradigma, percepções disparam ações. Por outro lado, pessoas e sistemas complexos de

robótica possuem medidas de percepção e ação distintas. Informação percebida chega através de diferentes sensores e em diferentes velocidades, freqüentemente de forma imprevisível.

O servidor utiliza um modelo de ação discreto: coleta ações de jogadores durante um ciclo fixo do simulador de tamanho *simulator_step*, mas apenas executa-as e atualiza o mundo no fim do ciclo. Se um cliente enviar mais de um comando de movimento num ciclo do simulador, o servidor escolhe um deles aleatoriamente para execução. Dessa forma, é do interesse de cada cliente tentar enviar pelo menos um comando de movimento a cada ciclo do simulador. Por outro lado, se um cliente não envia comando de movimento durante um ciclo, ele perde a oportunidade de agir durante aquele ciclo, o que pode ser uma significante desvantagem num domínio competitivo em tempo real: enquanto o agente permanece ocioso, oponentes podem obter vantagens. A cada ciclo, o simulador incrementa o contador de tempo em um.

O Soccer Server é uma simulação bi-dimensional. Não há noção de altura para qualquer objeto no campo. O campo possui as dimensões *comprimento x largura*, com gols de largura *g*, sendo estas dimensões parametrizáveis. Existem até 22 jogadores no campo, por vez, com 1 bola, os quais são modelados como círculos. Os jogadores e a bola são objetos móveis. No ciclo do simulador, no tempo *t*, cada objeto está numa posição específica (x,y) e possui uma velocidade específica *v*. Cada jogador está também olhando numa direção específica *d*. Estes dados são mantidos internamente.

O simulador adiciona ruído probabilisticamente distribuído para todos os objetos, o que de fato influencia na percepção de cada agente. Cada jogador possui uma quantidade de energia que ele poderá utilizar durante o jogo denominada *stamina*, que influencia na velocidade e resultado das diferentes jogadas efetuadas pelo agente. Também é possível a comunicação entre jogadores, que podem "falar" qualquer string de até 512 caracteres ASCII. Todos os jogadores, parceiros e oponentes, ouvirão a mensagem completa, segundo as restrições de faixa e freqüência do simulador.

Os agentes possuem atuadores para fisicamente manipular o mundo: virar-se, arrancar (correr) e chutar; o goleiro também pode agarrar a bola. Estes comandos são todos parametrizáveis, indicando ângulo e/ou força associada com cada ação, sendo que o servidor executa apenas um destes comandos por ciclo. Os jogadores ainda possuem comandos para controlar os tipos de percepção que receberão. O comando *sense_body* solicita ao servidor

informações físicas, enquanto que o comando *change_view* especifica uma qualidade de visão (alta/baixa) e um ângulo de visão (estreito/normal/largo). Alta qualidade e ângulo largo de visão levam a uma freqüência visual menor. *Sense_body* pode ser executado até três vezes por ciclo do simulador, enquanto que *change_view* pode ser executado apenas uma vez por ciclo.

Os dados relevantes das atividades dos jogadores, ou seja, posicionamento (do jogador e da bola), número do jogador, direção em que olha, modo de jogo e o próprio ciclo (momento) de cada um destes detalhes são registrados pelo servidor de jogos (Soccer Server) no log binário do simulador para cada ciclo da partida, o qual foi posteriormente convertido para o formato ASCII.

3.2 Passos do Processo

Com o objetivo de obter conhecimento relevante a respeito do comportamento, jogadas, atitudes e peculiaridades dos jogadores e dos times, foram extraídos logs de partidas da RoboCup do Campeonato de 1999, utilizando-se para isto o Soccer Monitor (Figura 5), um software que a partir do log binário apresenta, através de interface gráfica, a partida em seu ambiente virtual, sendo possível visualizar o contexto de todos os jogadores e suas respectivas jogadas em campo. Este software também converte estes logs binários em padrão ASCII.

Estes logs em ASCII foram processados, utilizando-se programas escritos em Prolog e Java, com o intuito de gerar duas tabelas (no SGBD SQL Server) de estatísticas a partir dos mesmos:

- Tabela flat primitivo – constituído por estatísticas de granularidade mínima, ou seja, informações a respeito da ação e posição de cada jogador a cada ciclo do simulador.

- Tabela flat derivado – constituído por estatísticas de maior granularidade, demonstrando diferentes jogadas (passe, chute, gol, lateral, defesa, roubada de bola), bem como dados relevantes às mesmas (momento de início e término, jogadores envolvidos, posição relativa dos jogadores e da bola).

A partir destas tabelas (fatos) e das tabelas descritivas destes fatos (dimensões) foi construído um Data Warehouse com o OLAP Manager - SQL Server cujo modelo é posteriormente descrito (Figura 8).

Uma visão geral do processo é apresentada na Figura 6, cujo objetivo final é a obtenção de várias estruturas de conhecimento a respeito das características, comportamentos e jogadas efetuadas no domínio da Robocup.

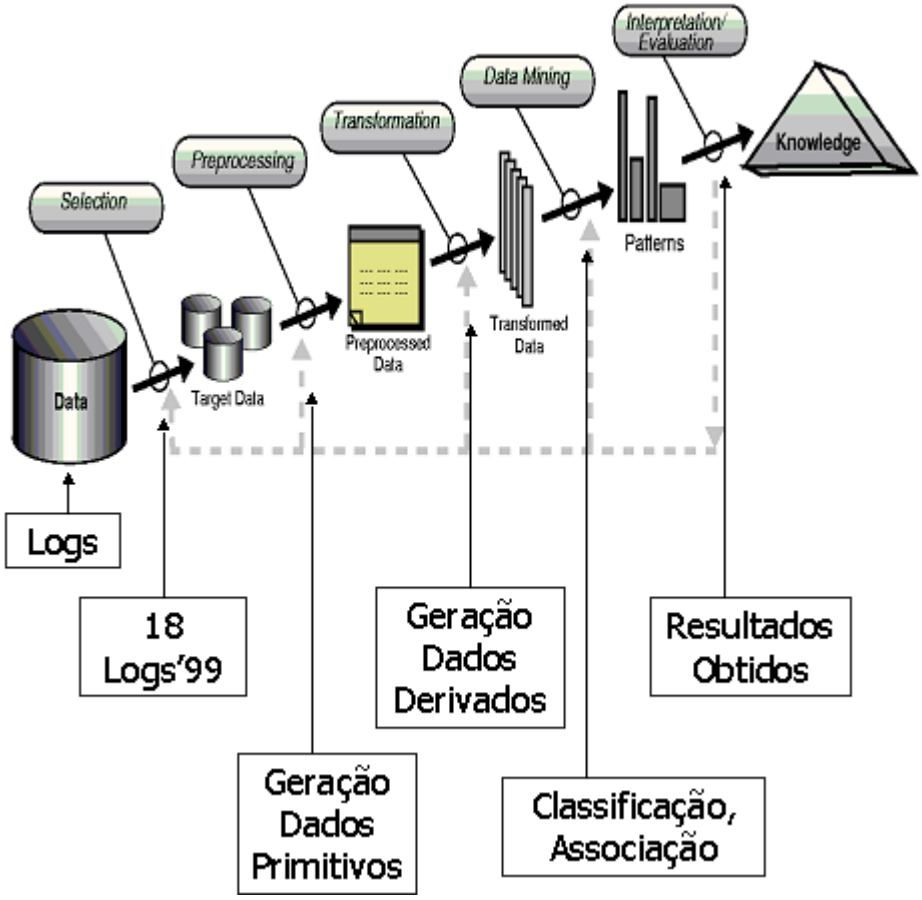

Figura 6: Passos do processo de KDD (Robocup)

3.2.1 Ferramentas Utilizadas

A escolha das ferramentas foi motivada pela necessidade de implementar, modelar e analisar a base de dados sob diferentes ângulos, visando a descoberta de conhecimento tático a respeito do comportamento, jogadas e particularidades dos times da RoboCup permitindo assim, por exemplo, prototipagem rápida de times competitivos para este domínio.

3.2.1.1 Weka

Waikato Environment for Knowledge Analysis (WEKA) - ferramenta de KDD que contempla uma série de algoritmos de aprendizagem de máquina - está descrita no capítulo de Contexto da Dissertação.

3.2.1.2 Microsoft SQL Server

O SQL Server [MIC2002] é o Sistema Gerenciador de Banco de Dados Relacional produzido pela Microsoft, também descrito no capítulo de Contexto da Dissertação.

3.2.1.3 Demais Ferramentas

- **Visio**

O Visio [MIC2002a] é uma ferramenta stand–alone da Microsoft, componente do Microsoft Office, voltada para a modelagem, manutenção e documentação de sistemas, bancos de dados, negócios, entre outros. Possui geração automática de tabelas a partir do modelo gráfico da modelagem. Este software foi utilizado para a modelagem do banco de dados da RoboCup.

- **XSB Prolog**

XSB Prolog [XSB2001] é uma linguagem de programação e um sistema de banco de dados dedutivo, capaz de resolver consultas recursivas. Caracteristicamente, realiza operações de *backtracking*, unificação de cláusulas e possui recursos para representação do conhecimento. Em relação a outras implementações de Prolog, o XSB apresenta-se mais seguro, pois é difícil fazer com que seu mecanismo de inferência entre em loop. Além disso, sua semântica de negação é muito bem fundamentada, tornando-o superior a outros Prologs no que diz respeito a regras com negação. XSB efetua cache de resultados e computações intermediárias de inferência, conferindo-lhe também eficiência além de segurança, principalmente em relação a loop e negação. No banco de dados RoboCup, o XSB Prolog foi utilizado para efetuar deduções sobre os logs ASCII e sobre os dados primitivos, gerando assim dados derivados semanticamente mais ricos. A escolha da versão XSB também foi influenciada pelo seu maior número de funcionalidades, API's e ferramentas do que as versões padrão de Prolog.

- **DBMiner**

Outra abordagem na análise dos dados foi a utilização do DBMiner (anteriormente citado) - ferramenta de mineração de dados desenvolvida pela DBMiner Technology Inc [DBM2000]. Esta ferramenta realiza tarefas de classificação, associação, clustering, entre outras, além de fornecer recursos para a visualização de resultados. Todas estas tarefas foram efetuadas sobre os cubos de dados (Primitivo e Derivado). Esta foi uma das razões que incentivaram a transformação da base de dados e a modelagem de um data warehouse com todas as informações disponíveis, pois o DBMiner de fato comporta-se como uma camada superior do Analysis Manager do SQL Server. Diferentemente do Weka, o DBMiner realizou suas tarefas diretamente sobre os cubos de dados gerenciados pelo Analysis Manager.

- **Progol**

Progol [ROB1997] é um ambiente de aprendizagem de máquina baseado na programação em lógica indutiva (ILP – Inductive Logic Programming). ILP é uma área de pesquisa formada pela interseção da Aprendizagem de Máquina com a Programação em Lógica, objetivando desenvolver descrições de predicados a partir de exemplos e conhecimento prévio. Não obstante, ILP é baseada na teoria da prova e na teoria dos modelos para o cálculo de predicados de primeira ordem. No caso de regras de classificação Id3, por exemplo, o resultado da aprendizagem realizada são regras específicas (proposicionais), enquanto que com ILP obtem-se regras genéricas (representação de primeira ordem). Desta forma, Progol efetua aprendizagem sobre exemplos que lhe são submetidos, gerando regras sobre determinado domínio. No banco de dados RoboCup, Progol foi utilizado de forma paralela às demais ferramentas para análise, visando a extração de conhecimento sobre os dados primitivos e derivados, com posterior avaliação e comparação entre o conhecimento gerado em formato de regras específicas e em formato de regras genéricas.

- **Java**

A linguagem de programação Java [SUN2001] foi utilizada para a manipulação de arquivos de dados e tabelas do banco de dados via JDBC, na implementação de JB2P(abordado adiante). Os programas recebiam e preparavam dados para o Prolog. Após receber o resultado da dedução do próprio Prolog, os programas Java armazenavam estes dados, conforme descrito na seção de Implementação do Aplicativo em Java e Prolog.

3.2.2 Seleção de Dados

Para a realização das tarefas ora descritas, foram selecionados 18 jogos de futebol de robôs da RoboCup da Simulation League – 1999.

Uma vez que a geração destes logs é realizada de forma automática pelo Soccer Server da RoboCup, os mesmos não apresentavam inconsistências, ausências ou outros problemas que demandassem limpeza destes dados, o que é bastante característico das bases de dados geradas automaticamente por software.

3.2.3 Implementação do Gerador de Dados Derivados em Java e Prolog

O gerador [ROC2001a] criado processa arquivos gerados pelos jogos da RoboCup (log files), produzindo tabelas e arquivos estruturados que podem ser usados para extrair conhecimento.

O principal problema quando se quer trabalhar com a RoboCup é a falta de padronização. Existem hoje soccer servers gerando arquivos em formatos completamente diferentes e com pouca ou nenhuma documentação. Então a primeira tarefa foi identificar um soccer server que executasse no Windows e que gerasse um arquivo de log em um formato compreensível. O aplicativo escolhido foi o soccer server monitor desenvolvido por Klaus Dorer [DOR2000].

O formato ASCII do arquivo de log gerado pelo soccer monitor é o seguinte:

```
Teamleft:FCPortugal,Teamright:LuckyLuebeck
Score:0:0
T:0,mode:4Ball:(0.0,0.0),P:1(-45.0,0.0,0),2(-11.0,18.0,81),3(-12.0,7.0,0)...
```

A primeira linha do arquivo contém o nome dos times que estão jogando; no exemplo acima *FCPortugal* e *LuckyLuebeck*. Essa é a primeira linha do arquivo que aparece uma única vez. O restante do arquivo é formado por uma linha que contém o placar. Exemplo: *Score:0:0* . A seguir, figura a linha que indica o tempo, o modo de jogo naquele momento e a posição dos objetos (bola e jogadores). Exemplo*: T:0,mode:4Ball:(0.0,0.0),P:1(-45.0,0.0,0),2(-11.0,18.0,81),3(-12.0,7.0,0)*, onde *T* indica o tempo 0 (inicial), *mode* indica o modo de jogo, *Ball* indica a posição da bola e *P* indica a posição dos jogadores (1, 2, 3...). A posição do jogador é formada por três campos que são (x, y, direção), onde a direção é um ângulo que indica para onde o jogador está olhando.

O gerador é constituído de duas partes: uma em Java e outra em Prolog. Java foi escolhida como linguagem de desenvolvimento devido à disponibilidade de componentes, interfaces e API's entre a linguagem e os demais elementos do gerador (Prolog, Weka, SQL Server). O programa Java é responsável por ler os dados e prepará-los para o Prolog, realizando as manipulações com arquivos e banco de dados.

O JB2P (Java Bridge to Prolog) [ROC2001] faz a ponte entre o programa em Java e o Prolog (XSB): um processo externo XSB é inicializado, permitindo que o programa em Java repasse ao XSB dados para dedução; o Prolog por sua vez processa dados recebidos gerando (derivando) novas informações. Após receber os resultados do Prolog, o programa em Java novamente manipula estes dados e utiliza JDBC [SUN2002] para o armazenamento dos dados primitivos (detalhados, com baixa granularidade) e derivados (dados sobre jogadas, em maior granularidade) em tabelas do SGBD (Figura 7). Caso seja necessário minerar diretamente arquivos flat com Weka (sem utilizar o SGBD), tais arquivos podem também ser gerados pela ferramenta.

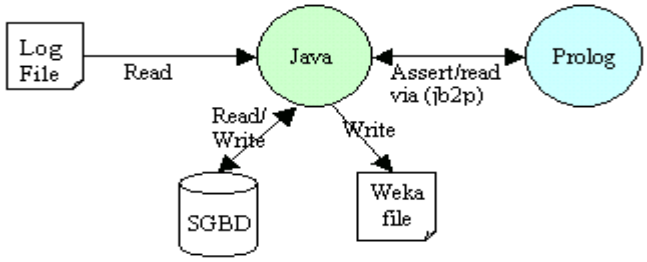

Figura 7: Aplicativo Java-Prolog

Os fatos primitivos em Prolog têm o seguinte formato:

```
fgame('FCPortugal','LuckyLuebeck').
fscore(0,0).
fmode(0,4).
fball(0,0.0,0.0).
fplayer(0,1,1,-45.0,0.0,0).
fplayer(0,1,2,-11.0,18.0,81)...
```

Onde: fplayer(0(tempo), 1(time), 2(número), -11.0(x), 18.0(y), 81(direção)).

Os fatos derivados em Prolog têm o seguinte formato (por exemplo, para posse de bola e para chute a gol):

```
ballPoss(1, 15, 30, 2.3, 5.4, 3.5, 6.8).
```

Onde: ballPoss(1(jogador), 15(tempo inicial), 30(tempo final), 2.3(coordenada X inicial da posição do jogador), 5.4(coordenada Y inicial), 3.5(X final), 6.8(Y final)).

```
shotAtGoal('blocked', 3, 1, 39, 43, 11.0, 6.8, 11.9, 7.3, 36., 27.2, 36.4, 26.9)
```

Onde: shotAtGoal('blocked' (resultado do chute: gol, defesa ou tiro de meta), 3 (jogador), 1 (time do jogador), 39 (momento inicial - chute), 43 (momento final - resultado), 11.0 (X inicial do jogador), 6.8 (Y inicial do jogador), 11.9 (X final do jogador), 7.3 (Y final do jogador), 36.9 (X inicial do goleiro), 27.2 (Y inicial do goleiro), 36.4 (X final do goleiro), 26.9(Y final do goleiro)).

As regras de derivação em Prolog para posse de bola e chute a gol (defendido) são:

```
ballPoss(Possessor, StartTime, EndTime, FromX, FromY, ToX, ToY) :-
        inicioDaPosse(StartTime, Time, Jogador, FromX, FromY, _),
        perdaDaPosse(StartTime, EndTime, Time, Jogador, ToX, ToY, _),
        jogadorAbsoluto(Possessor, Time, Jogador).


shotAtGoal(Type, Scorer, ScorerTeam, ShotStartTime, ShotEndTime, ShotFromX,
            ShotFromY, EndOfShotX, EndOfShotY, GoalieFromX, GoalieFromY, GoalieToX,
            GoalieToY) :-
defense(Scorer, ScorerTeam, ShotStartTime, ShotEndTime, ShotFromX, ShotFromY,
        EndOfShotX, EndOfShotY, GoalieFromX, GoalieFromY),
advTeam(AdvTeam, ScorerTeam),
fplayer(ShotEndTime, AdvTeam, 1, GoalieToX, GoalieToY, _),
Type = 'blocked'.
```

3.2.4 Implementação do Modelo de Dados

3.2.4.1 Modelagem

A modelagem do cubo de dados apresentou desafios para a execução da tarefa: a baixa granularidade, a característica espaço-temporal do domínio e a conseqüente necessidade de superar estes pontos. A primeira conclusão foi a necessidade de utilizar derivação para aumentar a granularidade dos dados e enriquecê-los. A segunda foi a constatação de que um modelo constelação para o cubo de dados (modelagem de várias tabelas de fato em um único

data warehouse com dimensões compartilhadas, formando uma "constelação de estrelas") [PET1999] atenderia melhor ao problema em questão.

A primeira tabela de fatos é a **Primitive_Flat** que contém a atividade e a posição absoluta de cada jogador em cada momento e de forma detalhada, numa granularidade mínima, a qual contém os seguintes campos:

Dimensões:

Game – Número do jogo (tabela Dim_Game)
Time – Tempo, cada ciclo do jogo (tabela Dim_Time)
Mode – Modo de jogo daquele momento – bola em jogo, impedimento, saída de bola, cobrança de lateral, escanteio ou tiro de meta, gol, final jogo (tabela Dim_Mode)
BallPosition – Posição da bola - (tabela Dim_RelCoords)
PlayerPosition – Posição do jogador (tabela_DimRelCoords)
Player – Número do jogador (tabela Dim_Agent)
Direction – Direção para a qual o jogador está olhando (tabela Dim_Direction)

Medidas:

AddTeam1Score – Incremento no placar - time 1
AddTeam2Score – Incremento no placar - time 2
Team1KickOff – Chute na bola para fora do campo – time 1
Team2KickOff – Chute na bola para fora do campo – time 2
Team1KickIn – Chute livre – time 1
Team2KickIn – Chute livre – time 2
Team1GoalKick – Chute a gol – time 1
Team2GoalKick – Chute a gol – time 2
Team1OffSide – Saída de bola – time 1
Team2OffSide- Saída de bola – time 2
Ocurrence – Ocorrência do registro na tabela. Sempre igual a 1 (campo auxiliar)

A segunda tabela de fatos é a **Derived_Flat**, que contém jogadas em granularidade maior, onde são descritos lances de posse de bola, passe, roubada de bola, saída de bola, chute a gol, defesa e gol. Seus campos são os seguintes:

Dimensões:

Game – Número do jogo (tabela Dim_Game)
Statistic – Tipo de jogada – gol, passe, roubada de bola, chute a gol, lateral, escanteio, tiro de meta, impedimento (Tabela Dim_Statistics)
Agent1 – Número do primeiro jogador envolvido na jogada (Tabela Dim_Agent)
Agent2 - Número do segundo jogador envolvido na jogada – caso ocorra (Dim_Agent)
StartTime – Momento do início da jogada (Tabela Dim_Time)
EndTime – Momento do término da jogada (Tabela Dim_Time)
BallFromTeam1 –Posição bola no início da jogada em relação ao time1 (Dim_RelCoords)
BallToTeam1 – Posição da bola no término da jogada em relação ao time 1
BallFromTeam2 - Posição da bola no início da jogada em relação ao time 2
BallToTeam2 – Posição da bola no término da jogada em relação ao time 2

Agent1FromTeam1 – Posição do agente 1 no início da jogada em relação ao time 1
Agent1ToTeam1 – Posição do agente 1 no término da jogada em relação ao time 1
Agent1FromTeam2 – Posição do agente 1 no início da jogada em relação ao time 2
Agent1ToTeam2 – Posição do agente 1 no término da jogada em relação ao time 2
Agent2FromTeam1 – Posição do agente 2 no início da jogada em relação ao time 1
Agent2ToTeam1 - Posição do agente 2 no término da jogada em relação ao time 1
Agent2FromTeam2 – Posição do agente 2 no início da jogada em relação ao time 2
Agent2ToTeam2 – Posição do agente 2 no término da jogada em relação ao time 2

Medidas:

DeltaTime – Tempo decorrido entre o início e o término da jogada
DeltaBallX – Distância percorrida pela bola durante a jogada em relação ao eixo X
DeltaBallY – Distância percorrida pela bola durante a jogada em relação ao eixo Y
DeltaAgent1X - Distância percorrida pelo agente 1 em relação ao eixo X
DeltaAgent1Y - Distância percorrida pelo agente 1 em relação ao eixo Y
DeltaAgent2X - Distância percorrida pelo agente 2 em relação ao eixo X
DeltaAgent2Y - Distância percorrida pelo agente 2 em relação ao eixo Y
Occurrence - Ocorrência do registro na tabela. Sempre igual a 1 (campo auxiliar)

As tabelas de dimensões são as seguintes:

Dim_Direction – Contém todas as direções possíveis para as quais o jogador pode olhar. Contém os seguintes campos:

Id_Direction – identificador da direção
AngleCoarse – direção cardeal (norte, sul, leste, oeste, nordeste, noroeste, sudoeste, sudeste)
AngleFine – ângulo exato da direção (-180 a +180)

Dim_Mode – Contém o modo de jogo. Campos:

Id_Mode – identificador do modo
ModeType – tipo (código do simulador) do modo - bola em jogo, impedimento, saída de bola, cobrança de lateral, escanteio ou tiro de meta, gol, final jogo.
Description – descrição do modo por extenso

Dim_RelCoords- Contém as coordenadas de posicionamento dos jogadores e da bola (Figuras 9, 10 e 11). Campos:

Id_RelCoords – identificador da coordenada
AreaCoarse – subdivisão das coordenadas em grandes quadrantes (Figura 9)
AreaMedium - subdivisão das coordenadas em médios quadrantes (Figura 10)
AreaFine - subdivisão das coordenadas em pequenos quadrantes (Figura 11)
X – Coordenada X
Y – Coordenada Y

Dim_Agent – Contém a identificação de cada jogador. Campos:

Id_Agent – identificador do agente (1 a 22)
Team – time do agente (1 ou 2)
Number – número do agente (1 a 11)

Dim_Time – Contém a subdivisão dos tempos do jogo. Campos:

Id_Time – identificador do tempo

Half – identificação de metade de tempo total de jogo (primeiro tempo, segundo tempo, tempo extra)
PeriodCoarse – identificação de quarto de tempo total de jogo
PeriodFine – identificação de oitavo de tempo total de jogo

Dim_Game – Contém a identificação dos times. Campos:

Id_Game – identificador do jogo
Game – nome do log do jogo
Team1 – nome do primeiro time
Team2 – nome do segundo time
Championship – nome do campeonato

Dim_Statistic – Contém a descrição de cada estatística (jogada) possível. Campos:

Id_Statistic – identificador da estatística
Type – tipo da estatística - bola em jogo, impedimento, saída de bola, cobrança de lateral, escanteio ou tiro de meta, chute, final de jogo
Subtype – subtipo da estatística - chute (a gol, para fora, defendido), saída de bola (lateral, escanteio, tiro de meta)
Description - descrição da estatística

O modelo de dados adotado encontra-se ilustrado na Figura 8, tendo sido implementado utilizando o OLAP Manager – SQL Server.

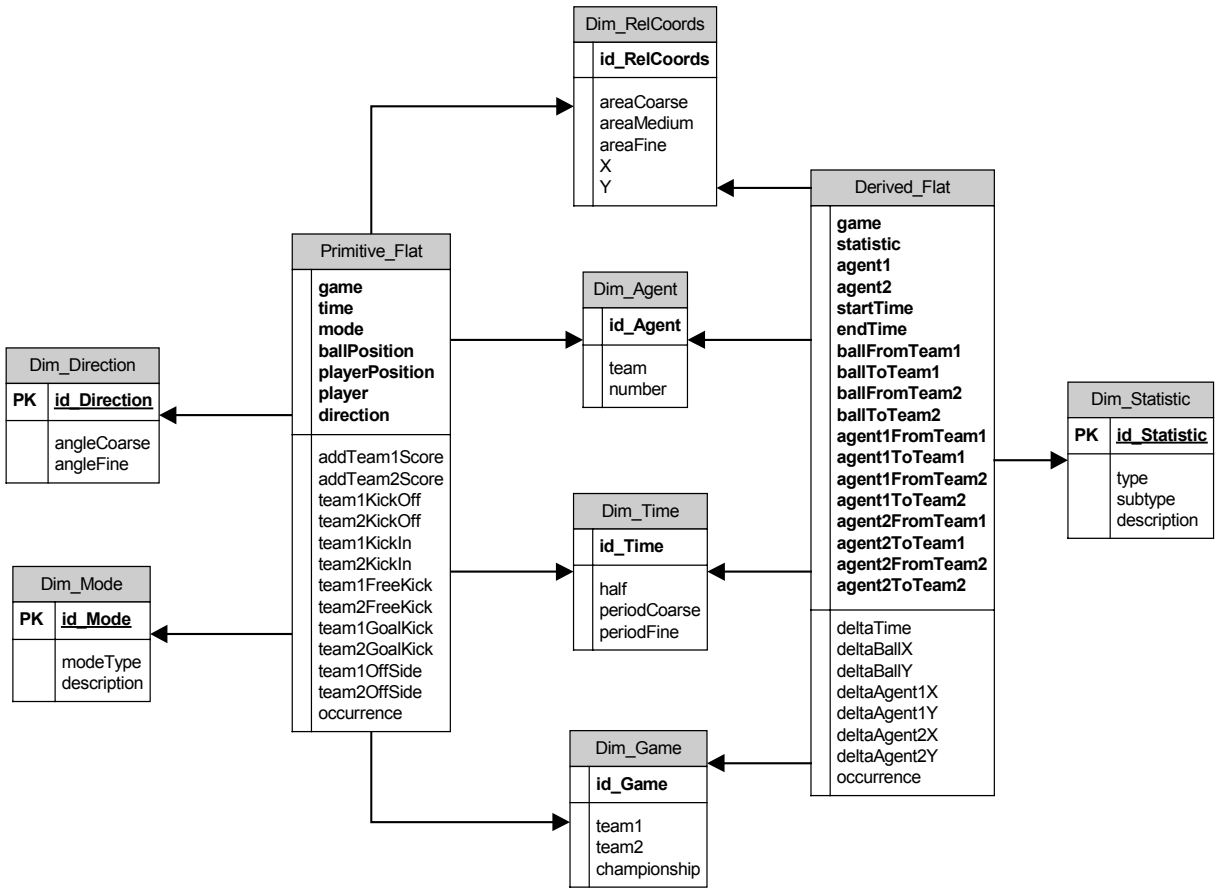

Figura 8: Modelo de dados RoboCup

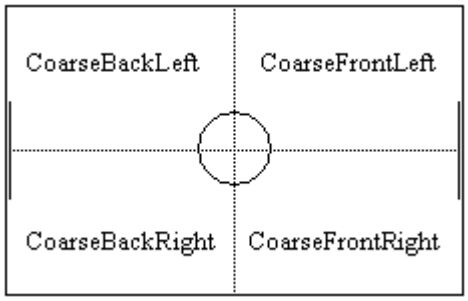

Figura 9: Áreas Coarse da dimensão Dim_RelCoords

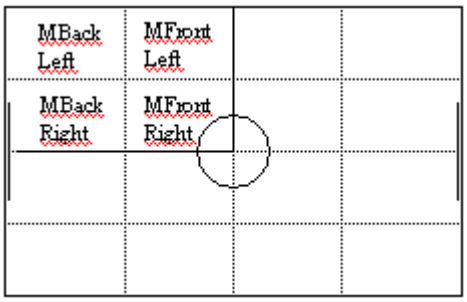

Figura 10: Áreas Medium da dimensão Dim_RelCoords

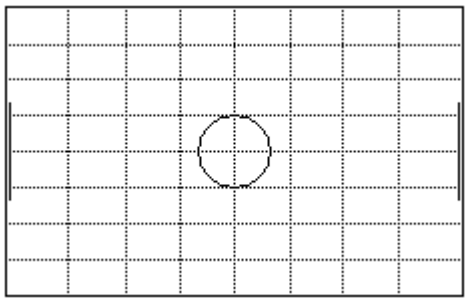

Figura 11: Áreas Fine da dimensão Dim_RelCoords

3.3 Resultados

3.3.1 Conhecimento descoberto

Através da comparação dos diferentes resultados obtidos, verificou-se que muitos pontos descobertos eram confirmados por mais de uma ferramenta de mineração utilizada, gerando assim resultados significativos abaixo apresentados.

3.3.1.1 Jogadas

- Das 16 áreas Medium, a maior proporção de **jogadas** realizadas encontra-se em três: CbrMfl (15,52%), CfrMbl (11,60%) e CfrMfl (10,81%). Juntas, estas áreas acumulam 37,93% de todas as jogadas realizadas, indicando um maior fluxo de jogadores no "corredor" formado por estas áreas (figura 12). Taticamente em relação à defesa, observa-se a demanda por uma atenção maior nesta região, seja através de marcação de jogadores ou ocupação deste espaço tão disputado. No que diz respeito ao ataque, este tipo de ação deve evitar esta região de "congestionamento", o que facilitará a condução e passe de bola.

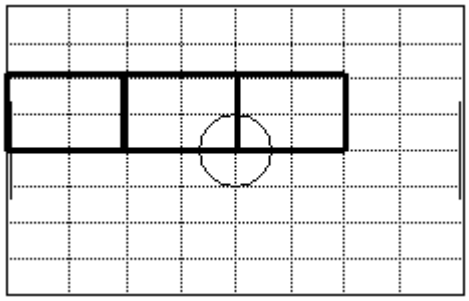

Figura 12: Corredor formado por CbrMfl, CfrMbl e CfrMfl

- Em relação à **troca de passes** foram verificadas as seguintes tendências:

  - 35,82% dos passes realizados ocorrem na mesma área Medium, enquanto que os demais, ou seja, 64,18% são realizados com lançador e lançado em quadrantes distintos;

  - enquanto que a maior proporção destes passes ocorre quando o lançador está em CbrMfl (18,76%), no caso do lançado este percentual atinge 15,99%, ocorrendo também em CbrMfl, conforme indicado na figura 13;

  - Estrategicamente, verifica-se duas diretrizes para marcação da troca de passes do adversário: (a) os passes são realizados predominantemente entre jogadores em posições distintas no que diz respeito à área Medium, ou seja, a interceptação de lançamentos deve considerar predominantemente distâncias médias; (b) Devido à freqüente utilização do "corredor" anteriormente referido, uma de suas áreas

(CbrMfl) figura como origem freqüente de lançamentos, confirmando o conhecimento descoberto.

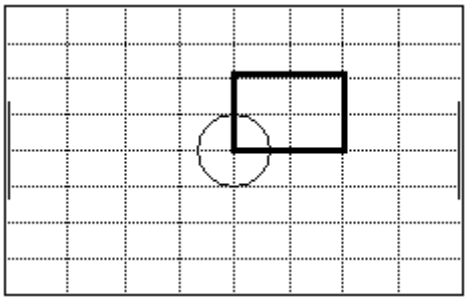

Figura 13: Área CbrMfl

3.3.1.2 Chutes e Gols

- Para os **chutes a gol** (em relação à posição do chutador) a maior frequência localiza-se em: CfrMfl (28,33%) e CflMfr (20,44%). Juntas, estas áreas totalizam 48,77% das conclusões de ataque. Do ponto de vista estratégico, fica confirmada a eminência geral do futebol do chute a gol nas imediações da grande área (figura 14).

- Em relação aos **gols** marcados, em relação à posição do goleador, verifica-se que ocorrem mais em CfrMfl (32,29%) e CflMfr (25,11%), totalizando assim 57,40% nestas áreas. Isto confirma (figura 14) a informação anterior e, do ponto de vista da defesa, reforça a necessidade de concentração de esforços no intuito de evitar e barrar chutes nesta região. Do ponto de vista do ataque, a estratégia de chutar de longe perde um pouco de consistência, embora não deva ser descartada.

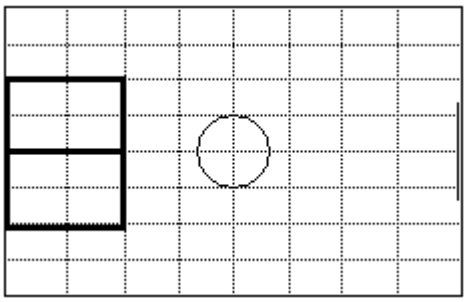

Figura 14: Áreas CfrMfl e CflMfr

- No que diz respeito ao **posicionamento do goleador e do goleiro**, verificou-se o seguinte (figura 15):

- Dos **gols** marcados, 58,74% ocorrem com o **goleador e o goleiro** na mesma área Medium;

- Em 34,53% dos **gols**, o **goleiro** estava em CfrMfl, e em 32,29% dos **gols**, o **goleador** estava em CfrMfl;

- A maior proporção de **gols** quando **ambos** encontram-se na mesma área Medium é de 23,77% ocorrendo em CfrMfl;

- Estrategicamente, estas informações contextualizadas com as anteriores indicam que: (a) o fato de permitir que o atacante adversário avance em direção ao gol obrigando o goleiro a "sair" aumenta consideravelmente as chances de gol do oponente; (b) a tendência de utilização do "corredor" continua implicando em finalizações com gol no seu segmento final (CfrMfl).

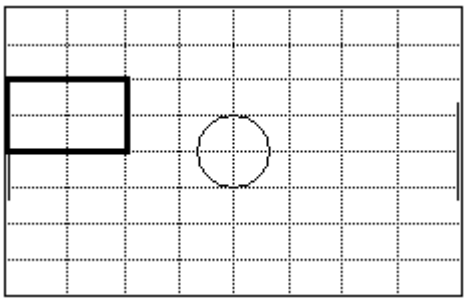

Figura 15: Área CfrMfl

3.3.2 Reflexão sobre o Processo

Em relação às ferramentas, verificou-se que a arquitetura do Weka possibilitou uma integração maior de recursos, permitindo acessos externos às suas diversas funcionalidades. Já o DBMiner, por sua característica de software fechado, limitou as tarefas aos recursos do seu ambiente, não permitindo extensibilidade de funções. Este software, como ferramenta de mineração, possui uma boa interface para o usuário, mas apresentou algumas falhas de funcionalidade; entretanto a mais contundente foi a impossibilidade de ajustar o tamanho dos conjuntos de dados de treinamento e de teste nas tarefas de classificação.

Consultas para derivação elaboradas em SQL (visando a comparação de resultados e de desempenho) apresentaram grande lentidão no SQL Server, sendo que algumas não puderam sequer ser executadas devido a limitações do SGBD. O InterProlog [DEC2001],

ferramenta utilizada a princípio para conexão Java-Prolog, mostrou-se ineficaz para esta aplicação, pois além de não possuir documentação mostrou-se sem escalabilidade para manipular uma grande quantidade de dados, apresentando travamentos durante a execução dos processos. Embora o XSB Prolog possua uma API ODBC, não foi possível utilizá-la devido a problemas de ambiente, conexão e permissões de acesso ao SGBD. Daí a necessidade de utilizar JB2P.

A utilização destes diferentes paradigmas e ferramentas viabilizou a implementação de aplicações voltadas a diferentes ambientes. Para essa aplicação, a utilização de Java permitiu via JDBC a manipulação de tabelas no SGBD SQL Server, uma vez que no Prolog XSB (via ODBC) esta funcionalidade foi inviabilizada devido a problemas de software. Outro fato relevante foi a utilização de Prolog para extrair conhecimento e produzir dados derivados dos logs da RoboCup que possuem granularidade muito pequena (dados primitivos).

Do ponto de vista global, a melhor combinação foi a utilização de: (a) SQL Server como gerenciador do repositório de dados; (b) Prolog como ferramenta de derivação de dados primitivos para obter dados derivados com mais semântica; (c) Weka para a realização das tarefas de mineração de dados, haja vista a diversidade de algoritmos e a arquitetura aberta do software que possibilita plena utilização de seus recursos.

O processo não trivial de integrar as três ferramentas de forma transparente, além da necessidade de especificar o processo de KDD de forma adequada e estruturada na combinação dos recursos disponíveis, reforçou a motivação para criação de uma linguagem de KDD, objetivando integrar estas ferramentas no protótipo inicial de implementação de SKDQL.

3.4 Conclusão

A complexidade espacial e temporal do domínio da RoboCup reafirmaram o grau de complexidade inerente às tarefas de KDD, bem como a demanda por métodos e ferramentas adequadas para descoberta de conhecimento em bancos de dados. O processo como um todo demandou bastante trabalho, levando a descobertas intrigantes confirmadas por diferentes ferramentas. Entretanto estas descobertas não são totalmente conclusivas para tomada de decisão sobre uma tática para time de futebol de robôs por serem dificilmente interpretáveis. Sendo assim, verifica-se que os dados são esparsos demais e de granularidade muito baixa

para viabilizar e constituir conclusões sólidas, necessitando-se de novos e sistemáticos passos para melhores resultados.

Este estudo de caso da RoboCup proporcionou relevante experiência da utilização de diferentes ferramentas e paradigmas para mineração de dados, bem como apontou para a necessidade de ambientes de KDD integrados e abertos. Conclui-se que uma linguagem com um conjunto de recursos e funcionalidades integrados, num ambiente que suporte a heterogeneidade de KDD, com diretivas para pré-processamento, mineração de dados e manipulação de conhecimento seria extremamente valiosa, flexibilizando e tornando mais eficiente todo o processo.

4 SKDQL – A Linguagem de Especificação

Uma característica desejável aos sistemas de KDD é a habilidade de suportar mineração ad hoc e interativa, com o intuito de flexibilizar e tornar mais eficiente a descoberta de conhecimento. Linguagens de KDD vêm ao encontro desta demanda quando fornecem diretivas que de forma transparente e integrada forneçam recursos e ferramentas comumente distintas e não integradas.

De acordo com a abordagem realizada, verifica-se que a descoberta de conhecimento em bancos de dados demanda usualmente uma série de tarefas de tratamento e análise de dados. Cada tarefa é composta por uma série de passos que efetuam, entre outros, seleção, limpeza, transformação e mineração de dados, além da apresentação e armazenamento de resultados e conhecimento. Com base neste fluxo natural de manipulação de dados e resultados, SKDQL possui cinco categorias de construtores apresentados adiante.

4.1 Escopo Atual da Especificação

A presente especificação de SKDQL (Structured Knowledge Discovery Query Language) define a sintaxe da linguagem estruturada para tarefas de descoberta de conhecimento que abrange:

- Recursos para acesso, recuperação e armazenamento de dados durante o processo de descoberta de conhecimento, herdando o poder e a funcionalidade das linguagens utilizadas nesta especificação, como por exemplo, SQL.

- Diretivas para pré-processamento dos dados selecionados, o que inclui limpeza, transformação, derivação e outros meios para enriquecer e aumentar a clareza, qualidade e representatividade dos dados disponíveis.

- Cláusulas que permitem a visualização, armazenamento e apresentação do conhecimento a priori ou resultante das tarefas de mineração em desenvolvimento.

- Algoritmos de mineração de dados para classificação, associação, clustering e relevância de atributos com diferentes opções e métodos de mineração, de teste, e de apresentação e armazenamento de resultados.

SKDQL aplica-se a tarefas de descoberta de conhecimento em bases e modelos de dados amplamente utilizados, fornecendo recursos relevantes a estas tarefas através de uma interface bastante difundida (semelhante a SQL). Uma vez que o objetivo da linguagem é a descrição de tarefas de KDD de forma iterativa e interativa, a sintaxe semelhante a SQL foi adotada por que além de bem conhecida, permite uma especificação declarativa, clara e direta de processos de KDD, herdando o poder e os recursos amplamente disponibilizados para esta sintaxe (em SGBD's, API's etc). Outras opções seriam:

- Interface baseada em linguagem de marcação (XML): embora este padrão tenha forte aceitação e ampla expansão nas aplicações para Web e intercâmbio de dados (uma das questões em KDD), a sua utilização como interface para o usuário dificultaria o fluxo de trabalho por se tratar de uma sintaxe verbosa, que requer constantemente marcadores nas suas cláusulas, aumentando consideravelmente o comprimento da especificação e requerendo do usuário extrema atenção e conhecimento deste domínio; entretanto, a utilização de XML para o intercâmbio de dados e representação de conhecimento é um dos pontos abordados para trabalhos futuros.

- Interface baseada em linguagem orientada a objetos: as características do paradigma OO, tais como herança e encapsulamento, oferecem recursos poderosos para aplicações computacionais. Contudo, para a especificação de processos, a utilização deste tipo de sintaxe tornaria menos clara e direta a especificação de processos de KDD. Todavia, o paradigma é utilizado na implementação de SKDQL através de Java.

- Interface baseada em linguagem dedutiva orientada a objetos: semelhantemente ao paradigma OO, esta sintaxe fornece recursos poderosos para a dedução e derivação de dados. No entanto, a sua utilização como sintaxe para a especificação de processos em descoberta de conhecimento esbarra em pontos semelhantes ao paradigma OO, além de exigir do usuário um prévio conhecimento de lógica dedutiva para a apropriada utilização de uma interface neste nível.

Assim como outras linguagens, SKDQL (mesmo especificada numa interface declarativa e de amplo domínio) pode ser mal utilizada, não sendo possível evitar que alguém defina e execute tarefas ambíguas ou impróprias, gerando resultados inconsistentes ou indesejados durante seu uso em tarefas de KDD.

4.1.1 Convenções da Especificação

A sintaxe de SKDQL foi expressa em EBNF (Extended Backus-Naur Form) [ISO1996]. Cabem aqui duas considerações em relação à especificação da linguagem. A primeira é que quando tratar-se de um símbolo terminal, a notação adotada define o seguinte: caso o símbolo seja iniciado com letra minúscula, este deve ser interpretado como um token (palavra reservada), como por exemplo em *dataSet*. Caso seja iniciado com letra maiúscula, o símbolo deve ser encarado como um identificador (de nome de arquivo, de tabela etc), como por exemplo em *DataSet*, que aguarda neste ponto o nome identificando o conjunto de dados.

O segundo ponto diz respeito ao conjunto de dados padrão, ou seja, quando não for explicitado por uma cláusula *from dataSet DataSet*, o conjunto de dados a ser considerado será o padrão, no qual são refletidas todas as operações realizadas e não especificadas pela cláusula *store in Dataset*. Tal recurso justifica-se pela agilidade proporcionada quando o usuário efetua operações sobre um conjunto de dados único (padrão) não sendo necessário estar especificando origem e destino de suas ações. Por outro lado, em outras ocasiões, certamente será necessário manipular diferentes bases de dados dentro da mesma operação de SKDQL, sendo fundamental a indicação dos dados nos quais a tarefa será realizada, bem como a definição do conjunto de dados destino desta tarefa.

4.2 Construtores para Especificação do Fluxo de um Processo de Descoberta de Conhecimento em Bancos de Dados

O símbolo inicial da linguagem é o não-terminal <**SKDQLtask**>, que define recursivamente uma tarefa como uma seqüência de passos de tratamento de dados (SKDQLstep). É definido da sequinte forma:

```
<SKDQLtask> ::= <SKDQLstep> {<Conj> <SKDQLtask>}
```

onde <Conj> pode ser o terminal “and” ou “then”, dependendo da semântica que se deseja representar entre as tarefas realizadas (seqüenciadas ou paralelizadas):

```
<Conj> ::= then | and
```

<**SKDQLstep**> é definido como um passo de preparação de dados (Prepare), seguido por uma atividade opcional de conhecimento prévio (PriorKnowledge), tendo logo após um passo de mineração de dados (Mine) com subseqüente apresentação de resultados (Present) para interpretação e avaliação.

A iteratividade e a interatividade característica dos processos de KDD apontaram para a necessidade de flexibilização, alternância e combinação destas tarefas, embora ancoradas na seqüência natural do processo de descoberta de conhecimento (Figura 1). Por exemplo, passos alternados de preparação e mineração sobre uma base de dados podem ser necessários durante uma determinada fase do trabalho para atingir o objetivo da tarefa proposta. Daí, <SKDQLstep> ser definido também, individualmente, como passo de preparação, pré-processamento, conhecimento a priori, mineração e apresentação, visando a flexibilização e dinamização do processo como um todo:

```
<SKDQLstep> ::=
   <Prepare> <Conj> [<PriorKnowledge> <Conj>] <Mine> [<Conj> <Present>] |
   <Prepare>                                                             |
   <Preprocess>                                                          |
   <PriorKnowledge>                                                      |
   <Mine>                                                                |
   <Present>
```

<SKDQLtask> e <SKDQLstep> configuram-se nas cláusulas de mais alto nível de SKDQL. A partir delas, teremos as demais cláusulas responsáveis por diferentes formas de tratamento de dados em KDD.

4.3 Construtores para Especificação de Acesso e Armazenamento de Dados durante um Processo de Descoberta de Conhecimento em Bancos de Dados

A cláusula <**Prepare**> possui duas sub-cláusulas a serem consideradas, <Pick> e <Preprocess>:

```
<Prepare> ::= <Pick> {<Conj> <Preprocess>}
```

Analisaremos, então, a funcionalidade de cada uma delas, destacando suas características e recursos.

4.3.1 A cláusula <Pick>

O passo inicial numa tarefa de KDD é a especificação do conjunto de dados a ser explorado durante todo o processo. Este passo demanda a especificação da fonte dos dados (servidores, bancos de dados, data warehouses etc), além de recursos para uma seleção de dados mais peculiar (amostragem, por exemplo). A cláusula <Pick> suporta a conexão e seleção de dados para os passos subseqüentes, sendo composta por: <ConnectTo> e, opcionalmente, <BDQuery> e <Select>:

```
<Pick> ::= <ConnectTo> [<Conj> <BDQuery>]
           {<Conj> <Select>} [store in DataSet]
```

4.3.2 A Cláusula <ConnectTo>

<ConnectTo> fornece parâmetros básicos para a conexão à fonte de dados – modelo de dados, url, instância da base dos dados, login e password:

```
<ConnectTo> ::= connectTo <SourceModel> at <SourceLoc>

<SourceModel> := relationalDB     |
                 dataCube         |
                 semiStructuredDB |
                 flatFile

<SourceLoc> := Url, DataBase, Login, Password
```

Exemplo: `connectTo relationalDB at Cin03, Robocup, robocup, robocup`

4.3.3 A Cláusula <BDQuery>

<BDQuery> possui recursos para consultas padrão a diferentes modelos de dados (consultas SQL, MDX - cubos OLAP, XML), inclusive carga de arquivos flat. Além disso, sua sintaxe permite que a consulta realizada seja opcionalmente armazenada num dataset para posterior manipulação, seja num pré-processamento, mineração ou apresentação:

```
<BDQuery> ::= <Query> [store in DataSet]

<Query> ::= <SQLquery>     |
            <MDXquery>     |
            <Xquery>       |
            <FlatQuery>

<FlatQuery> ::= readFlatFile FileName
```

Exemplo:

```
select distinct F.game, F.time, F.player, RB.x, RB.y,
       RP.x, RP.y
from   primitive_flat F, dim_relcoords RB,
       dim_relcoords RP
where  F.ballPosition = RB.id_relcoords and
       F.playerPosition = RP.id_relcoords and
       game = 1 and time < 200
```

4.3.4 A Cláusula <Select>

A cláusula opcional <**Select**> possui duas funcionalidades úteis às tarefas de KDD, sendo a primeira a seleção de atributos segundo critérios estabelecidos e a segunda a amostragem de instâncias através de parâmetros específicos:

```
<Select> ::= <SelectFeature> |
             <Sample>
```

<**SelectFeature**> permite a remoção ou generalização de atributos segundo sua sintaxe:

```
<SelectFeature>::= removeSubsumedFeatures  with threshold = A      |
                   generalizeFeature with Attribute threshold = A  |
                   generalizeFeature with relation threshold = R
```

`removeSubsumedFeatures` é um símbolo terminal para a funcionalidade que determina a exclusão de atributos que (1) possuam um número de valores distintos superior ao parâmetro *threshold* e (2) não possuam nenhum operador de generalização de hierarquia de conceitos [HAN2001]. Esta abordagem é justificada pelo seguinte raciocínio: um grande número de valores distintos em um atributo (sem o recurso de um conceito de hierarquia definido) quando removido elimina também restrições, generalizando regras que utilizem esta base de dados. A sua manutenção implicaria num número maior de disjunções, que contradiz a intenção de gerar regras concisas.

`generalizeFeatureWith` determina (1) a generalização de um atributo *Attribute* enquanto este possuir um número de valores distintos superior ao parâmetro *threshold* ou (2) a generalização de atributos (que possuam tal operador) até que toda a relação fique limitada ao parâmetro *threshold* informado.

4.3.5 A Cláusula <Sample>

<**Sample**> permite a realização de amostragem nos dados segundo os parâmetros informados:

```
<Sample> ::= sample <Qty> from <SamplingSource> using <SamplingPolicy>

<Qty> ::= integer           |
          integerPercentage

<SamplingSource> ::= allData              |
                     <BinningMethod> bins

<BinningMethod> ::= equiwidth   |
                    equidepth   |
                    vOptimal    |
                    maxDiff boundary = B

<SamplingPolicy> ::= randomWithoutReplacement |
                     randomWithReplacement
```

<**Qty**> informa a quantidade ou percentual (inteiros) do tamanho da amostra a ser realizada. `<SamplingSource>` determina se a amostra será realizada sobre todo o conjunto de dados ou se a amostragem será feita sobre bins (subconjuntos) gerados pelos seguintes métodos (<**BinningMethod**>):

- `equiwidth` - o tamanho de cada subconjunto é uniforme.

- `equidepth` - a freqüência de cada subconjunto é constante.
- `V-Optimal` – a variância de cada subconjunto é a menor possível.
- `MaxDiff` – a diferença entre cada par de valores adjacentes é tida como um limite (B) estabelecido para a formação do subconjunto.

<**SamplingPolicy**> determina se a amostragem será realizada com ou sem reposição de valores.

Exemplo:

```
then sample % 50 from alldata using randomWithoutReplacement
```

4.4 Construtores para Especificação de Tarefas de Pré-Processamento de Dados

Dentro da cláusula `<Prepare>`, logo após a cláusua `<Pick>` segue-se `<Preprocess>`, que será analisada a partir deste ponto.

4.4.1 A cláusula <Preprocess>

<**Preprocess**> contempla as tarefas de limpeza, transformação, derivação, mistura aleatória e recuperação de dados. Algumas das suas opções contemplam os recursos *from dataSet DataSet* e *store in DataSet*, conforme sua sintaxe:

```
<Preprocess> ::= <Clean>          |
                 <Transform>      |
                 <Derive>         |
                 <Randomize>      |
                 <RestoreDataSet>
```

4.4.2 A Cláusula <Clean>

Dados oriundos de aplicativos, de bases de dados, da Web e demais fontes do mundo real tendem a ser incompletos, ruidosos, redundantes e inconsistentes. A limpeza de dados em KDD permite que estas distorções sejam identificadas e corrigidas.

<**Clean**> envolve um conjunto de funcionalidades pertinentes à limpeza de dados através de diferentes recursos. Primeiramente, segue abaixo a sintaxe desta cláusula e de suas sub-cláusulas:

```
<Clean> ::= [from dataSet DataSet]
            clean <FeatureSpec> using <CleaningMethod>
            [store in DataSet]

<FeatureSpec> ::= allFeatures   |
                  TargetFeature

<CleaningMethod> ::= <FillNull>                |
                     <Smooth>                  |
                     removeWithMissingValues

<FillNull> ::= defaultValue = V   |
               defaultString = S  |
               <AvgAggregFunction> [withRespectTo AggregFeature]

<AvgAggregFunction> ::= mean               |
                        median             |
                        mostFrequent       |
                        <TrimmedMean>

<TrimmedMean> ::= trimmedMean excluding <Qty>

<Smooth> ::= <SmoothingMethod> from <BinningMethod> bins

<SmoothingMethod> ::= means       |
                      boundaries
```

<**FeatureSpec**> determina se a tarefa de limpeza de dados será realizada sobre todos os atributos, ou se contemplará determinado atributo.

<**CleaningMethod**> permite que seja escolhido o método de limpeza aplicado, a saber:

- <**FillNull**> indica que o(s) atributo(s) nulos devem ser preenchidos segundo uma das seguintes diretivas:
    - `defaultValue = V` - ser preenchido com um valor informado.
    - `defaultString = S` – ser preenchido com uma string informada.
    - `<AvgAggregFunction> [withRespectTo AggregFeature]` – ser preenchido por uma função (<**AvgAggregFunction**>) de agregação (média, mediana, mais freqüente, *trimmed mean* – média preparada) sobre determinado atributo (`AggregFeature`) das instâncias.

- <**TrimmedMean**> - *trimmed mean* (média preparada) é calculada através do descarte de uma quantidade ou percentual das faixas inferiores e superiores de valores considerada, calculando-se então a média dos valores restantes.

- <**Smooth**> - indica que o atributo deve ser nivelado (removido seu ruído) segundo as seguintes diretivas:

  - <**SmoothingMethod**> :

    - means – cada valor do bin (subconjunto) deve ser substitituído pelo valor médio deste bin.

    - Boundaries – os valores máximos e mínimos de cada bin substituem os demais, prevalecendo aquele (valor máximo ou mínimo) que estiver mais próximo do valor a ser substituído.

  - <**BinningMethod**> - método (equiwidth, equidepth, v-optimal e maxDiff) que irá gerar os bins para a tarefa de smoothing.

- `removeWithMissingValues` – remover as instâncias cujo(s) atributo(s) sejam nulos.

Exemplo: `then from dataset area.dat clean 4 using removeWithMissingValues store in area1.dat`

4.4.3 A Cláusula <Transform>

Os dados a serem minerados nem sempre apresentam-se no formato apropriado. Provenientes muitas vezes de ambientes distintos, verifica-se a necessidade de transformá-los visando uma manipulação mais efetiva dos mesmos.

<**Transform**> realiza duas transformações de dados (normalização e compressão) segundo a sintaxe:

```
<Transform> ::= <Normalize>  |
                <Compress>

<Normalize> ::= [from dataSet DataSet]
                normalize <NormFeatureSpec> by <NormalizeMethod>
                [store in DataSet]
```

```
<NormFeatureSpec> ::= allNumFeatures   |
                      TargetFeature

<NormalizeMethod> ::=  minMax within MinValue MaxValue     |
                       zScore                              |
                       decimalScaling


<Compress> ::= [from dataSet DataSet]
               compressDownTo <Qty> using <CompressMethod>
               [store in DataSet]

<CompressMethod> ::= <WaveletMethod> |
                     pca

<WaveletMethod> ::= haar2       |
                    daubechies4 |
                    daubechies6
```

<**Normalize**> efetua normalização em um atributo específico ou sobre todos os atributos numéricos, podendo utilizar os seguintes métodos (<NormalizeMethod>):

- `minMax` – que define a faixa de valores (MinValue, MaxValue) na qual a normalização deve ser realizada.
- `zScore` – normalização baseada na média e no desvio padrão do atributo.
- `decimalScaling` – normaliza através da movimentação do ponto decimal do valor do atributo, tal que um valor *v* do atributo é: *Max (|v'|) < 1*.

<**Compress**> - permite a compressão de dados, visando a obtenção de uma representação reduzida dos dados originais. São disponibilizados dois métodos (<**CompressMethod**>):

- <**WaveletMethod**> - baseado na transformada wavelet discreta – DWT (uma técnica de processamento linear de sinais que quando aplicada a um vetor de dados *D*, transforma-o num vetor *D'* numericamente diferente, de coeficientes wavelet). As transformadas *Haar_2*, *Daubechies_4 e Daubechies_6* são as mais utilizadas [ROW2001].
- **PCA** – Principal Components Analysis, conhecido por *Karhunen-Loeve ou K-L method* [STA2002]. O método realiza a busca por vetores ortogonais k-dimensionais que possam representar os dados da melhor forma possível.

Assim, os dados originais são projetados num espaço muito menor, resultando na compressão dos dados a partir da redução de dimensionalidade.

4.4.4 A Cláusula <Derive>

O enriquecimento de dados é um recurso utilizado para melhorar a semântica da base a ser minerada, pois muitas vezes a forma primitiva dos dados apresenta-se pobre sob este aspecto. A derivação de dados é um método muito útil para aumentar a expressividade destes dados.

<**Derive**> possibilita que derivações de dados sejam realizadas, utilizando extensão de esquema de dados relacionais, através de queries SQL e regras Prolog [XSB2001], segundo a sintaxe:

```
<Derive> ::= <DeriveElements>  {<DeriveElements>} |
             <DeriveHierarchy>

<DeriveElements> ::=
              <SQLSchemaExtend> <DeriveElements>   |
              derive using <BDquery>               |
              derive from <BDquery> applying Rules from
              PrologRules store in DataSet

<DeriveHierarchy> ::= <Discretize>  |
                      <Generalize>

<Discretize> ::= [from dataSet DataSet]
                 discretize <FeatureSpec> using <DiscretizeMethod>
                 [store in DataSet]

<DiscretizeMethod> ::= <AvgAggregFunction> from <BinningMethod> bins |
                       entropy                                       |
                       naturalPartitioning excluding <Qty>           |
                       partitioning in N bins

<Generalize> ::= [from dataSet DataSet]
                 generalize TargetFeature using <AvgAggregFunction>
                 [withRespectTo AggregFeature]
                 [store in DataSet]
```

<**DeriveElements**> determina a derivação de elementos, ou seja, a manipulação de atributos e instâncias de uma base de dados para realizar três tipos de tarefas:

- extensão de esquema de dados relacional.
- derivação de dados via consultas SQL.

- derivação de dados através da submissão de uma base de dados (obtida através de consulta) a determinadas regras Prolog, armazenadas em um arquivo de regras *PrologRules* com o armazenamento do resultado da derivação em *DataSet*.

<**DeriveHierarchy**> permite a discretização e generalização de atributos. Ambos os métodos permitem uma redução racional do número de valores distintos de um atributo, especialmente se este for contínuo (discretização). Isto é especialmente benéfico para métodos de mineração que são tipicamente recursivos, pois um número elevado de valores distintos tende a aumentar o tempo total de processamento destes métodos.

A discretização é definida da seguinte forma:

- <**Discretize**> atua sobre um ou todos os atributos utilizando os seguintes métodos de discretização:

  - função de agregação sobre bins – neste caso, os bins (vizinhos com valores aproximados) são gerados segundo o método indicado e, nestes subconjuntos, a função de agregação especificada é aplicada para a obtenção do valor que irá substituir os valores originais deste bin.

  - entropia – uma medida baseada no ganho de informação e na distribuição de classes das amostras no conjunto denominada entropia [HIL2001] é utilizada para particionar recursivamente os valores do atributo.

  - particionamento natural – uma regra 3-4-5 é utilizada para segmentar dados numéricos recursivamente em intervalos relativamente uniformes e naturais, baseando-se na faixa de valores do dígito mais significativo [HAN2001]. Desta forma, através deste método, serão gerados intervalos de tamanhos equivalentes de forma automática e uniforme.

  - particionamento sobre bins – método de particionamento que utiliza MDL (Minimum Description Length) [FAY1993], que prioriza a solução mais simples possível. Neste caso, o conjunto é particionado igualitariamente em N bins, conforme especificado pelo usuário.

- <**Generalize**> determina que o atributo seja generalizado segundo uma das funções de agregação <AvgAggregFunction>: média, mediana, elemento mais freqüente e média preparada – trimmed mean (onde o cálculo da média é realizado após o descarte de um percentual superior e inferior do intervalo considerado).

Exemplo:

```
then derive from
     prologQuery
            select distinct F.game, F.time, F.player, RB.x, RB.y,
                   RP.x, RP.y
        from   primitive_flat F, dim_relcoords RB, dim_relcoords RP
        where  F.ballPosition = RB.id_relcoords and
               F.playerPosition = RP.id_relcoords and
               game = 1 and time < 200
     prologQuery
     applying rulePosse.R
     from testePosse.P
     store in tposse
```

4.4.5 As Cláusulas <Randomize> e <RestoreDataSet>

<**Randomize**> basicamente realiza um embaralhamento na ordem das instâncias da base de dados. Tal procedimento é útil uma vez que alguns algoritmos de mineração podem gerar resultados enviesados devido à ordenação ou outra particular disposição das instâncias de dados. Eis sua sintaxe:

```
<Randomize> ::= [from dataSet DataSet]
                randomize elements [store in DataSet]
```

As tarefas de KDD manipulam intensamente a base de dados tratada, provocando constantes modificações na mesma. Caso uma instância desta base tenha sido armazenada em algum momento deste processamento, é possível recuperá-la e torná-la o dataset padrão através da cláusula <**RestoreDataSet**>, que possui a seguinte sintaxe:

```
<RestoreDataSet> ::= restore dataset DataSet
```

4.5 Construtores para Especificação de Tarefas de Apresentação do Conhecimento

O conhecimento prévio de determinado domínio pode indicar caminhos e soluções que efetivamente aumentem a qualidade ou agilidade do processo de descoberta de conhecimento. Pode, inclusive, modificar totalmente a abordagem realizada sobre determinada massa de dados. Por isso, SKDQL possui cláusulas para especificar o conhecimento prévio e apresentar o conhecimento obtido dentro da tarefa em andamento.

4.5.1 A cláusula <PriorKnowledge>

<**PriorKnowledge**> possui recursos para acessar uma base de dados, para verificar uma amostragem de um conjunto de dados e para definir a priori a formação de regras de associação, segundo sua sintaxe:

```
<PriorKnowledge> ::= <ConnectTo> <Conj> <BDQuery> |
                     <ViewSampleOfDataset>        |
                     <AssociationPriorKnowledge>  |
                     <Present>

<ViewSampleOfDataset> ::= viewSampleOf "%" N dataSet DataSet

<AssociationPriorKnowledge> ::= matching <MetaRule>

<MetaRule> ::= Term {<Operator> Term} imply Term {<Operator> Term}

<Operator> ::= and | or
```

Conforme sua sintaxe, <PriorKnowledge> utiliza a estrutura anteriormente descrita para conectar-se a uma base de dados, e realizar consultas sobre ela. Além disso, é possível também visualizar uma amostragem de um dataset anteriormente armazenado através da cláusula <**ViewSampleOfDataset**>, onde é fornecido o percentual do tamanho da amostra a ser visualizada.

<**AssociationPriorKnowledge**> permite a definição de uma meta-regra que determinará a formação das regras de associação a serem obtidas durante este processo de mineração, onde *Term* representa predicados da meta-regra.

4.5.2 A Cláusula <Present>

<**Present**> possibilita a visualização de informações que tenham sido previamente armazenados através da cláusula <**Display**>, bem como a junção de diferentes arquivos deste tipo em apenas um, utilizando para isto a cláusula <**JoinDisplay**>.

```
<Present> ::= <Display>    |
              <JoinDisplay>

<Display> ::= display FileName

<JoinDisplay> ::= joinDisplay FileName {FileName} store in FileName
```

4.6 Construtores para Especificação de Tarefas de Mineração de Dados

SKDQL fornece recursos para a mineração de vários tipos de conhecimento (ou modelos) através de diferentes métodos e algoritmos, com meios para validação, teste e outras opções. A mineração de atributos relevantes, de modelos de classificação, de regras de associação e de clustering estão contemplados nesta especificação [WAI2001].

4.6.1 A Cláusula <Mine>

<**Mine**> é definida a partir da seguinte sintaxe:

```
<Mine> ::= <MineRelevantAttributes> |
           <MineClassification>     |
           <MineAssociations>       |
           <MineClusters>
```

4.6.2 A Cláusula <MineRelevantAttributes>

Dentro de um conjunto de dados, encontram-se atributos que têm um peso maior nas tarefas de mineração de dados. Por exemplo, no momento de classificar um bom ou mal pagador de um empréstimo bancário, com certeza a renda do indivíduo será mais relevante do que a sua naturalidade. Eles são chamados de atributos relevantes, que podem ser minerados através da cláusula <**MineRelevantAttributes**>:

```
<MineRelevantAttributes> ::= mineRelevantAttributes [analyze Feature]
                             [from dataSet DataSet]
                             [storeRelevantAttributes in FileName]
                             [apply filter and store in DataSet]
                             using <AttributeEvaluator>
                             [testOptions <RelevanceTestOptions>]

<AttributeEvaluator> ::= infoGain                                      |
                         gainRatio                                     |
                         correlationSubsetEval searchMethod <SMethod>  |
                         consistencySubsetEval searchMethod <SMethod>

<SearchMethod> ::= BestFirst |
                   Genetic   |
                   Random    |
                   Exaustive

<RelevanceTestOptions> ::= useFullTrainingSet |
                           crossValidation NumFolds
```

A mineração de atributos relevantes deve ser feita em relação a um dos atributos do conjunto de dados, denominado classe. `[analyze Feature]` determina a classe da tarefa de mineração e, uma vez que é opcional, pode ser omitido fazendo com que seja assumido o default que é eleger o último atributo selecionado da base de dados como classe.

A tarefa de mineração será desenvolvida sobre o conjunto de dados especificado na cláusula `[from dataSet DataSet]`, e caso seja omitida, o conjunto de dados considerado será o último pré-processado (dataset padrão).

Além da obtenção dos atributos relevantes, é possível filtrar com `apply filter` o conjunto de dados para que sejam eliminados os demais atributos e mantidos somente os relevantes e o atributo classe para as tarefas subseqüentes. A informação dos atributos relevantes pode ser armazenada com `[storeRelevantAttributes in FileName]`. O conjunto de dados resultante pode ser então armazenado com `store in DataSet`.

A mineração de atributos relevantes deve ser realizada através de um avaliador de atributos, definido em `using <`**`AttributeEvaluator`**`>` [WIT2000]:

- infoGain – avalia atributos individualmente através da medição de ganho de informação com respeito à classe, ou seja, para cada atributo é medida a quantidade de ganho de informação para a classe em questão.

- gainRatio – realiza avaliação de atributos individualmente através da medição de relação de ganho com respeito à classe.

- correlationSubsetEval – avalia o valor de um subconjunto de atributos considerando a habilidade individual preditiva de cada atributo, juntamente com o grau de redundância entre eles.

- consistencySubsetEval – desenvolve a avaliação do valor de um subconjunto de atributos através do nível de consistência nos valores da classe quando as instâncias de treinamento são projetadas sobre o subconjunto de atributos.

Os avaliadores *correlationSubsetEval* e *consistencySubsetEval* podem utilizar diferentes métodos de busca (que realizarão a pesquisa no subespaço do atributo), determinados pela cláusula <**SearchMethod**>:

- BestFirst – faz busca no espaço de subconjuntos de atributos através do método guloso hillclimbing, acrescido de uma facilidade de backtracking.

- Genetic – desenvolve a busca utilizando um algoritmo genético simples.

- Random – realiza uma busca aleatória no espaço de subconjuntos de atributos.

- Exaustive – faz uma busca exaustiva no espaço de subconjuntos de atributos a partir do conjunto vazio.

A mineração de atributos relevantes ainda possui opções de testes (para os diferentes avaliadores e métodos de busca) presentes na cláusula <**RelevanceTestOptions**>:

- useFullTrainingSet – determina que todo o conjunto dados seja utilizado para o treinamento.

- crossValidation – permite que seja informado o número de folds para realização de Cross Validation, onde cada um destes folds (subconjuntos) de dados é utilizado para teste e o restante para treinamento.

Exemplo:

```
then mineRelevantAttributes
        from dataSet data1.dat
        apply filter and store in data2.dat
        using correlationSubsetEval searchMethod BestFirst
```

4.6.3 A Cláusula <MineClassification>

Os métodos de classificação em mineração de dados são utilizados para determinar e avaliar modelos que forneçam recursos para classificar ou prever determinado objeto ou evento. A sintaxe das tarefas de classificação são determinadas por <**MineClassification**>, que possui a seguinte sintaxe:

```
<MineClassification>::= mineClassification [analyze Feature]
                        [from dataSet DataSet]
                        [present <ClassPresentation> [store in FileName]]
                        [storeAllPresentations in FileName]
                        classifier <ClassAlgorithm>
                        [testOptions <ClassTestOptions>]

<ClassPresentation> ::= model             |
                        evaluationSummary |
                        confusionMatrix

<ClassAlgorithm> ::= decisionTable |
                     id3           |
                     naiveBayes    |
                     prism         |
                     j48 [confidenceFactor = Value]

<ClassTestOptions> ::= useTrainingSet            |
                       suppliedTestSet <BDQuery> |
                       crossValidation NumFolds  |
                       percentageSplit Percentage
```

A classificação deve, assim como a relevância de atributos, ser realizada em relação a um atributo denominado classe, possuindo também como default o último atributo do conjunto de dados.

A tarefa de mineração será desenvolvida sobre o conjunto de dados especificado na cláusula `[from dataSet DataSet]`, e caso seja omitida, o conjunto de dados considerado será o último pré-processado (dataset padrão).

O *modelo* gerado, o *resumo da avaliação* do modelo ou a *matriz de confusão* podem ser apresentados ao usuário e, opcionalmente, armazenados individualmente em arquivo, segundo `[present <ClassPresentation> [store in FileName]]`. Também opcionalmente todos estes podem ser armazenados em arquivo em um único passo com `[storeAllPresentations in FileName]`.

A cláusula <**ClassAlgorithm**> disponibiliza cinco algoritmos de classificação [WIT2000]:

- Decision Table – produz uma tabela de decisão simples.
- ID3 – gera árvore de decisão id3.
- Naive Bayes – método simplista que assume independência de eventos.
- Prism – algoritmo que utiliza a indução de regras modulares.
- J48 – implementa árvores de decisão C4.5 .

Novas opções de teste para classificação estão especificadas em <**ClassTestOptions**>:

- `SuppliedTestSet` – permite que uma base de dados externa seja utilizada para teste do modelo.
- `PercentageSplit` – define o percentual da base de dados que será utilizada para treinamento, sendo o restante utilizado para teste.

Exemplo:

```
then mineClassification
           from dataSet data2.dat
           present model
           store in model1.skdql
           classifier id3
```

4.6.4 A Cláusula <MineAssociations>

Regras de associação são semelhantes a regras de classificação, sendo que elas podem prever qualquer atributo, e não apenas um dos atributos que deve ser pré-determinado (classe), permitindo assim que diferentes combinações de atributos sejam contempladas nestas regras. A sintaxe de <**MineAssociations**> é a seguinte:

```
<MineAssociations> ::= mineAssociations
                       [from dataSet DataSet]
                       [present rules [store in FileName]]
                       {with <AssocParameter> = Value}

<AssocParameter> ::= confidence |
                     support    |
                     numRules
```

Assim como nos outros métodos, a tarefa de mineração será desenvolvida sobre o conjunto de dados especificado na cláusula `[from dataSet DataSet]`, e caso seja omitida, o conjunto de dados considerado será o último pré-processado (dataset padrão). O algoritmo utilizado é o Apriori, o qual iterativamente reduz o suporte mínimo até que seja encontrado o número desejado de regras com a confidência mínima fornecida [AGR1994].

As regras geradas podem ser apresentadas e opcionalmente armazenadas através de `[present rules [store in FileName]]`. Os parâmetros que estabelecem confidência, suporte e número máximo de regras podem ser estabelecidos por `{with <AssocParameter> = Value}`.

Exemplo:
```
then mineAssociations
from dataSet data1.dat
present rules
store in assoc1.skdql
```

4.6.5 A Cláusula <MineClusters>

A mineração de clusters apresenta, segundo critérios pré-definidos, o formato de um diagrama que mostra como as instâncias de uma base de dados se distribuem em grupos (clusters). Abaixo é apresentada a sintaxe de <**`MineClusters`**>:

```
<MineClusters> ::= mineClusters [ignore Feature]
                   [from dataSet DataSet]
                   [present clusters]
                   [store in FileName]
                   clusterer <ClusterAlgorithm>
                   [testOptions <ClusterTestOptions>]

<ClusterAlgorithm> ::=  cobweb                                  |
                        simpleKMeans [numClusters = Value]      |
                        em [numClusters = value] [maxIterations = value]

<ClusterTestOptions> ::= useTrainingSet                         |
                         suppliedTestSet <BDQuery>              |
                         percentageSplit Percentage
```

Neste processo de mineração, é facultado ao usuário determinar um atributo a ser ignorado pelo algoritmo. Tal opção é necessária quando não for conveniente aplicar limpeza de dados sobre este atributo, delegando assim ao algoritmo de clustering a tarefa de ignorá-lo. Esta cláusula também possui `[from dataSet DataSet]`, com a mesma semântica dos demais métodos. A apresentação e armazenamento de resultados da mineração são analogamente fornecidos por `[present clusters]` e `[store in FileName]`.

Os algoritmos de clustering disponíveis em <**ClusterAlgorithm**> são Cobweb, Simple Kmeans e EM [WIT2000], com seus respectivos parâmetros. As opções para teste de <ClusterTestOptions> seguem o padrão definido para os demais métodos.

4.7 Especificação da Gramática de SKDQL

```
<SKDQLtask> ::= <SKDQLstep> {<Conj> <SKDQLtask>}

<SKDQLstep> ::=
   <Prepare> <Conj> [<PriorKnowledge> <Conj>] <Mine> [<Conj> <Present>] |
   <Prepare>                                                             |
   <Preprocess>                                                          |
   <PriorKnowledge>                                                      |
   <Mine>                                                                |
   <Present>

<Prepare> ::= <Pick> {<Conj> <Preprocess>}

<Conj> ::= then |
           and

<Pick> ::= <ConnectTo> [<Conj> <BDQuery>]
           {<Conj> <Select>} [store in DataSet]

<ConnectTo> ::= connectTo <SourceModel> at <SourceLoc>

<SourceModel> := relationalDB     |
                 dataCube         |
                 semiStructuredDB |
                 flatFile

<SourceLoc> := Url, Instance, Login, Password

<BDQuery> ::= <Query> [store in DataSet]

<Query> ::= <SQLquery>     |
            <MDXquery>     |
            <Xquery>       |
            <FlatQuery>

<FlatQuery> ::= readFlatFile FileName

<Select> ::= <SelectFeature> |
             <Sample>

<SelectFeature> ::= removeSubsumedFeatures with threshold = A      |
                    generalizeFeature with Attribute threshold = A |
                    generalizeFeature with relation threshold = R

<Sample> ::= sample <Qty> from <SamplingSource> using <SamplingPolicy>

<Qty> ::= integer           |
          integerPercentage

<SamplingSource> ::= allData             |
                     <BinningMethod> bins
```

```
<BinningMethod> ::= equiwidth   |
                    equidepth   |
                    vOptimal    |
                    maxDiff boundary = B


<SamplingPolicy> ::= randomWithoutReplacement |
                     randomWithReplacement

<Preprocess> ::= <Clean>          |
                 <Transform>      |
                 <Derive>         |
                 <Randomize>      |
                 <RestoreDataSet>

<Clean> ::= [from dataset DataSet]
            clean <FeatureSpec> using <CleaningMethod>
            [store in DataSet]

<FeatureSpec> ::= allFeatures   |
                  TargetFeature

<CleaningMethod> ::= <FillNull>             |
                     <Smooth>               |
                     removeWithMissingValues

<FillNull> ::= defaultValue = V   |
               defaultString = S  |
               <AvgAggregFunction> [withRespectTo AggregFeature]

<AvgAggregFunction> ::= mean               |
                        median             |
                        mostFrequent       |
                        <TrimmedMean>

<TrimmedMean> ::= trimmedMean excluding <Qty>

<Smooth> ::= <SmoothingMethod> from <BinningMethod> bins

<SmoothingMethod> ::= means       |
                      boundaries

<Transform> ::= <Normalize>  |
                <Compress>

<Normalize> ::= [from dataset DataSet]
                normalize <NormFeatureSpec> by <NormalizeMethod>
                [store in DataSet]

<NormFeatureSpec> ::= allNumFeatures   |
                      TargetFeature

<NormalizeMethod> ::=  minMax within MinValue MaxValue     |
                       zScore                              |
                       decimalScaling

<Compress> ::= [from dataset DataSet]
               compressDownTo <Qty> using <CompressMethod>
               [store in DataSet]
```

```
<CompressMethod> ::= <WaveletMethod> |
                     pca

<WaveletMethod> ::= haar2       |
                    daubechies4 |
                    daubechies6

<Derive> ::= <DeriveElements>  {<DeriveElements>} | <DeriveHierarchy>


<DeriveElements> ::= <SQLchemaExtend> <DeriveElements>   |
                     derive using <BDquery>              |
                     derive from <BDquery> applying Rules
                     from PrologRules store in DataSet

<DeriveHierarchy> ::= <Discretize>  |
                       <Generalize>

<Discretize> ::= [from dataset DataSet]
                 discretize <FeatureSpec> using <DiscretizeMethod>
                 [store in DataSet]

<DiscretizeMethod> ::= <AvgAggregFunction> from <BinningMethod> bins |
                       entropy                                       |
                       naturalPartitioning excluding <Qty>           |
                       partitioning in N bins

<Generalize> ::= [from dataset DataSet]
                 generalize TargetFeature using <AvgAggregFunction>
                 [withRespectTo AggregFeature]
                 [store in DataSet]

<Randomize> ::= [from dataset DataSet] randomize elements
                [store in DataSet]

<RestoreDataSet> ::= restore dataset DataSet

<PriorKnowledge> ::= <ConnectTo> <Conj> <BDQuery> |
                     <ViewSampleOfDataset>        |
                     <AssociationPriorKnowledge>  |
                     <Present>

<ViewSampleOfDataset> ::= viewSampleOf % N dataSet DataSet

<AssociationPriorKnowledge> ::= matching <MetaRule>

<MetaRule> ::= Term {<Operator> Term} imply Term {<Operator> Term}

<Operator> ::= and | or

<Present> ::= <Display>    |
              <JoinDisplay>

<Display> ::= display FileName

<JoinDisplay> ::= joinDisplay FileName {FileName} store in FileName

<Mine> ::= <MineRelevantAttributes> |
           <MineClassification>     |
           <MineAssociations>       |
           <MineClusters>
```

```
<MineRelevantAttributes> ::= mineRelevantAttributes [analyze Feature]
                        [from dataSet DataSet]
                        [storeRelevantAttributes in FileName]
                        [apply filter and store in DataSet]
                        using <AttributeEvaluator>
                        [testOptions <RelevanceTestOptions>]


<AttributeEvaluator> ::= infoGain                                      |
                         gainRatio                                     |
                         correlationSubsetEval searchMethod <SMethod>  |
                         consistencySubsetEval searchMethod <SMethod>

<SearchMethod> ::= BestFirst |
                   Genetic   |
                   Random    |
                   Exaustive

<RelevanceTestOptions> ::= useFullTrainingSet          |
                           crossValidation NumFolds

<MineClassification> ::=mineClassification [analyze Feature]
                         [from dataSet DataSet]
                         [present <ClassPresentation> [store in FileName]]
                         [storeAllPresentations in FileName]
                         classifier <ClassAlgorithm>
                         [testOptions <ClassTestOptions>]

<ClassPresentation> ::= model             |
                        evaluationSummary |
                        confusionMatrix

<ClassAlgorithm> ::=    decisionTable   |
                        id3             |
                        naiveBayes      |
                        prism           |
                        j48 [confidenceFactor = Value]

<ClassTestOptions> ::= useTrainingSet            |
                       suppliedTestSet <BDQuery> |
                       crossValidation NumFolds  |
                       percentageSplit Percentage

<MineAssociations> ::= mineAssociations
                       [from dataSet DataSet]
                       [present rules [store in FileName]]
                       {with <AssocParameter> = Value}

<AssocParameter> ::= confidence |
                     support    |
                     numRules

<MineClusters> ::= mineClusters [ignore Feature]
                   [from dataSet DataSet]
                   [present clusters]
                   [store in FileName]
                   clusterer <ClusterAlgorithm>
                   [testOptions <ClusterTestOptions>]
```

```
<ClusterAlgorithm> ::=  cobweb                                |
                        simpleKMeans [numClusters = Value]    |
                        em [numClusters = value] [maxIterations = value]

<ClusterTestOptions> ::= useTrainingSet                       |
                         suppliedTestSet <BDQuery>            |
                         percentageSplit Percentage
```

4.8 Exemplo de uso Prático: Processo de Mineração de Arquivos de Log da RoboCup em SKDQL

Para exemplificar a utilização prática de SKDQL, será retomado o problema da busca de padrões para melhorar a tática de times da RoboCup, através de uma seqüência de tarefas de KDD sobre este domínio. Para isto, supõe-se a necessidade de um usuário analisar o banco de dados de logs da RoboCup, objetivando obter conhecimento a respeito das situações e características das diferentes jogadas realizadas neste ambiente, utilizando para isto recursos como derivação, associação e classificação. Exemplos são apresentados a seguir:

- Neste primeiro exemplo, após a conexão ao banco de dados, através de uma consulta SQL uma base de dados é selecionada. Logo após é realizada uma amostragem de 50% sobre esta base, com a subseqüente aplicação de um algoritmo de classificação Naive Bayes. Esta tarefa objetiva a aquisição de conhecimento geral sobre a realidade da base de dados utilizando um classificador básico:

```
connectTo relationalDB at Cin03, Robocup, robocup, robocup

then sqlQuery
     select T.periodFine, C.areaFine, C.areaMedium, A.number,
          M.description
     from   Primitive_Flat F, Dim_Time T, Dim_Mode M,
          Dim_RelCoords C, Dim_Agent A
     where  F.time = T.id_Time  and
          F.mode = M.id_Mode  and
          F.ballPosition = C.id_RelCoords and
          F.playerPosition = C.id_RelCoords and
          F.player = A.id_Agent and
          F.game = 1 and F.time < 200
   sqlQuery

then sample % 50 from alldata using randomWithoutReplacement

then mineClassification
   present model
```

```
store in naive.skdql
classifier Id3
```

****

- Neste passo subseqüente, após a seleção da base de dados, uma tarefa de mineração é realizada utilizando-se a classificação com Id3. O modelo Naive Bayes anteriormente gerado é também apresentado. Neste ponto, o objetivo consiste em utilizar um algoritmo clássico para obter informações mais precisas visando a comparação entre uma amostra com Naive Bayes e o conjunto de dados completo com Id3:

```
connectTo relationalDB at Cin03, Robocup, robocup, robocup

then sqlQuery
    select T.periodFine, C.areaFine, C.areaMedium, A.number,
        M.description
    from   Primitive_Flat F, Dim_Time T, Dim_Mode M,
        Dim_RelCoords C, Dim_Agent A
    where  F.time = T.id_Time  and
        F.mode = M.id_Mode  and
        F.ballPosition = C.id_RelCoords and
        F.playerPosition = C.id_RelCoords and
        F.player = A.id_Agent and
        F.game = 1 and F.time < 200
  sqlQuery

then mineClassification
  present model
  classifier Id3

then display naive.skdql
```

****

- Neste ponto, após a conexão, os dados gerados por uma consulta são submetidos à regra *rulePosse.R* do banco de regras *testePosse.P*, sendo o resultado armazenado em *tposse*. Os dados derivados são então classificados por um algoritmo Decision Table. O intuito da tarefa é derivar dados primitivos (melhorando sua semântica) e aplicar neles um algoritmo de tabela de decisão para análise dos resultados e compará-los com resultados anteriores de mineração sobre dados puramente primitivos:

```
connectTo relationalDB at Cin03, Robocup, robocup, robocup
then derive from
  prologQuery
      select distinct F.game, F.time, F.player, RB.x, RB.y, RP.x, RP.y
```

```
    from   primitive_flat F, dim_relcoords RB, dim_relcoords RP
    where  F.ballPosition = RB.id_relcoords and
           F.playerPosition = RP.id_relcoords and
           game = 1 and time < 200
  prologQuery
  applying rulePosse.R
  from testePosse.P
  store in tposse

then mineClassification analyze 3
  from dataSet tposse.dat
  present model
  store in regras.skdql
  classifier decisionTable
```

****

- Na seleção realizada, é efetuada uma limpeza no atributo *number* e, com o intuito de verificar o efeito desta limpeza, são realizadas tarefas de mineração com o algoritmo *Prism* na base antes da limpeza e após a limpeza:

```
connectTo relationalDB at Cin03, Robocup, robocup, robocup

then sqlQuery
        select T.periodFine, C.areaFine, C.areaMedium, A.number,
          M.description
      from   Primitive_Flat F, Dim_Time T, Dim_Mode M,
          Dim_RelCoords C, Dim_Agent A
      where  F.time = T.id_Time  and
          F.mode = M.id_Mode  and
          F.ballPosition = C.id_RelCoords and
          F.playerPosition = C.id_RelCoords and
          F.player = A.id_Agent and
          F.game = 1 and F.time < 200
    sqlQuery
    store in area.dat

then from dataset area.dat clean 4 using removeWithMissingValues store in
area1.dat

then mineClassification
    from dataSet area.dat
    present model
    classifier prism

then mineClassification
    from dataSet area1.dat
    present model
    classifier prism
```

5 Implementação da Interface SKDQL do MATRIKS

SKDQL, cuja interface figura na arquitetura do MATRIKS, foi implementada através de um protótipo que possui as principais funcionalidades da linguagem. Algumas características desta implementação serão discutidas e descritas a seguir.

5.1 Plataforma de Implementação

Para implementação de SKDQL, optou-se pela plataforma Java devido aos seguintes fatores:

- A arquitetura do próprio MATRIKS já adota Java como plataforma padrão para o desenvolvimento de seus componentes.
- A plataforma Java, bem como interfaces desenvolvidas, possuem suporte para plataformas heterogêneas, muito comuns em KDD.
- Os padrões de distribuição, extensibilidade, interoperabilidade e encapsulamento de componentes adotados para o MATRIKS (numa perspectiva de sistema aberto e extensível) são contemplados na plataforma Java.

A disponibilidade de diversos componentes, ferramentas e recursos implementados em Java favorecem o desenvolvimento e a integração de novas funcionalidades à arquitetura através da reutilização destes componentes. Por outro lado, os componentes e funcionalidades do MATRIKS também podem ser reutilizados ou acessados por outro software.

5.2 Estrutura do Software

O acesso aos dados, o pré-processamento, a mineração de dados e a apresentação de resultados são requisitos fundamentais na implementação do protótipo de SKDQL. Seguindo a proposta de integração, extensibilidade e encapsulamento da arquitetura do MATRIKS, diferentes API's, componentes e softwares foram utilizados para a implementação destas funcionalidades conforme descrito a seguir e ilustrado na Figura 16.

Para o acesso a bancos de dados relacionais, foi utilizado JDBC [SUN2002], uma API que permite conexão a fontes de dados tabulares a partir de programas Java, provendo conectividade para um grande número de sistemas gerenciadores de bancos de dados que utilizam SQL. Para esta implementação, foi utilizado o Microsoft SQL Server como SGBD do repositório de dados.

Para funcionalidades de pré-processamento e mineração de dados foram utilizados componentes do WEKA [WAI2001], uma coleção de algoritmos de manipulação e mineração de dados (filtragem, normalização, classificação, associação, clustering etc.) escritos em Java e encapsulados em componentes de software que são chamados diretamente por SKDQL.

O Prolog XSB [XSB2001] é uma linguagem de programação e um sistema de banco de dados dedutivo capaz de resolver consultas recursivas, realizar operações de *backtracking* e unificação de cláusulas. Este software é utilizado nas operações de derivação de dados através de cláusulas SKDQL, sendo acessado através de JB2P [ROC2001], uma API que permite que programas Java realizem chamadas e obtenham resultados do Prolog XSB.

A apresentação do conhecimento foi realizada com operações de exibição de texto em tela (árvores, matrizes, regras de associação, de classificação etc.), sendo possível gravar em arquivo estes resultados de forma individual ou agrupados segundo especificações do usuário.

5.3 Ferramentas de Software

Para o desenvolvimento do parser de SKDQL, foi utilizado o gerador de parser JavaCC – Java Compiler Compiler 2.1 [WEB2002]. O gerador de parser lê uma especificação de gramática (padrão JavaCC) e a converte em um programa Java capaz de verificar a validade de sentenças para esta gramática (análise léxica e sintática). A versão de Java utilizada no desenvolvimento foi o JDK 1.3 [SUN2001].

Além da verificação de validade de sentenças da linguagem, através de JavaCC também é possível especificar chamadas a métodos e componentes Java diretamente na gramática padrão JavaCC. Desta forma foi implementada a geração de código descrita a seguir.

Para obter esta gramática padrão JavaCC, foi necessário redefinir toda a especificação apresentada da gramática neste pseudo-código padrão do gerador de parser, que por sua vez

foi submetido ao JavaCC. Este, além de gerar o parser (programas Java), também foi muito útil na depuração de SKDQL, uma vez que a análise desempenhada pelo mesmo apontava erros e inconsistências nas versões iniciais da linguagem.

5.4 Geração de Código

Para a geração de código de SKDQL, foram selecionadas funcionalidades de acesso, pré-processamento e mineração de dados, bem como de apresentação de resultados. De forma transparente via JDBC, o código de SKDQL gerado por JavaCC realiza o acesso aos dados relacionais, bastando informar URL, banco de dados, login e senha. A consulta em SQL é identificada por SKDQL e repassada para JDBC.

As tarefas de pré-processamento e de mineração são realizadas através de chamadas de métodos aos componentes do WEKA, os quais são escritos também em Java. Para acessar o Prolog, foi utilizado JB2P [ROC2001] que permite que programas Java façam chamadas ao Prolog para a execução de regras e posterior recebimento do resultado destas, implementando assim uma ponte Java-Prolog.

Numa sequência de execução de uma tarefa SKDQL, o protótipo gerado a partir da especificação da gramática padrão JavaCC acessa (segundo definido nas cláusulas SKDQL da tarefa proposta) a base de dados através do SGBD via JDBC, efetua tarefas de pré-processamento e mineração de dados através dos componentes Weka diretamente invocados via código Java e realiza operações de derivação de dados utilizando o PROLOG via JB2P (Figura 16). Finalmente, os resultados (conforme especificado na tarefa) são exibidos em tela e/ou armazenados em arquivo para análise do usuário. Estas operações possuem intercalações e seqüenciamentos opcionais, visando oferecer uma maior flexibilidade e poder ao usuário no desempenho de suas atividades de KDD.

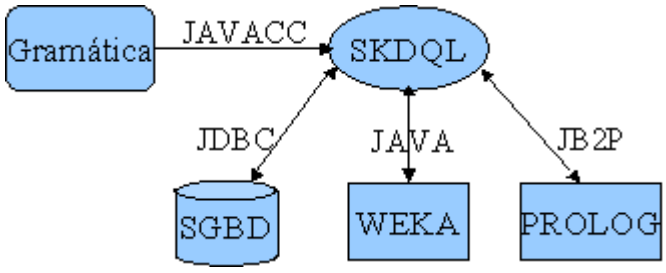

Figura 16: Esquema para Geração de Código

5.5 Exemplo de Uso

Para exemplificar a utilização de SKDQL, considere-se uma tarefa que consiste em analisar bases de dados originais, derivadas e filtradas através de algoritmos de associação e classificação. A série de passos requeridos e a diversidade de operações realizadas sobre os dados demandariam diferentes softwares e, conseqüentemente, diferentes questões relacionadas a estes softwares e seus ambientes (integração de ferramentas, configuração destes ambientes, manipulação de dados etc.) teriam de ser tratadas pelo usuário.

Problemas desta natureza geralmente consomem mais tempo e esforço do que a atividade de análise de dados em si. O Parser de SKDQL, através da integração de diferentes funcionalidades, abstrai o analista dessas questões permitindo-lhe especificar diretamente nas diretivas da linguagem as operações sobre os dados.

Utilizando o parser SKDQL para efetuar a tarefa proposta (e especificada abaixo em SKDQL), o primeiro passo consiste em conectar-se ao banco de dados e obter uma das bases de dados desejada, utilizando para isso uma consulta SQL, cujo resultado será armazenado num arquivo *data1.dat*. A seguir, é realizado um passo de derivação aplicando-se sobre o resultado de uma *query* a regra Prolog *posse* (armazenada em *rulePosse.R*) do banco de regras *testePosse.P* e armazenando os dados derivados em *tposse.dat.*

O próximo passo consiste em realizar a filtragem dos atributos relevantes de *data1.dat* utilizando o algoritmo *correlationSubsetEval* para esta tarefa. A base de dados filtrada é armazenada em *data2.dat*. Logo após, tarefas de classificação utilizando os algoritmos Id3 e j48 são realizadas sobre os dados filtrados. Em seguida uma tarefa de classificação com Id3 é aplicada sobre os dados não filtrados (*data1.dat*). Na seqüência, um algoritmo de classificação de Tabela de Decisão é aplicado sobre os dados derivados *tposse.dat*. Um algoritmo de associação também é aplicado sobre estes dados não filtrados (*data1.dat*). Os resultados de cada uma destas tarefas é armazenado em arquivo.

Finalizando, os resultados das tarefas de mineração, para efeitos de praticidade, são agrupados em um único arquivo (*result.skdql*), que é apresentado para análise do usuário e comentado adiante. Segue abaixo a descrição deste procedimento em SKDQL:

```
connectTo relationalDB at Cin03, Robocup, robocup, robocup

then sqlQuery
      select T.periodFine, C.areaFine, C.areaMedium, A.number, M.description
      from   Primitive_Flat F, Dim_Time T, Dim_Mode M,
             Dim_RelCoords C, Dim_Agent A
     where F.time = T.id_Time  and
           F.mode = M.id_Mode  and
           F.ballPosition = C.id_RelCoords and
           F.playerPosition = C.id_RelCoords and
           F.player = A.id_Agent and
          F.game = 1 and F.time < 200
   sqlQuery
   store in data1.dat

then derive from
   prologQuery
        select distinct F.game, F.time, F.player, RB.x, RB.y, RP.x, RP.y
       from   primitive_flat F, dim_relcoords RB, dim_relcoords RP
      where  F.ballPosition = RB.id_relcoords and
             F.playerPosition = RP.id_relcoords and
             game = 1 and time < 200
   prologQuery
   applying rulePosse.R
   from testePosse.P
   store in tposse

   then mineRelevantAttributes
       from dataSet data1.dat
       apply filter and store in data2.dat
       using correlationSubsetEval searchMethod BestFirst


          then mineClassification
                from dataSet data2.dat
                present model
               store in model1.skdql
               classifier id3

       and  mineClassification
             from dataSet data2.dat
              present model
             store in model2.skdql
             classifier j48

then mineClassification
   from dataSet data1.dat
   present model
   store in model3.skdql
   classifier id3
```

```
then mineClassification
    from dataSet data1.dat
    present confusionMatrix
    store in matId3.skdql
    classifier id3

then mineClassification analyze 3
    from dataSet tposse.dat
    present model
    store in model4.skdql
    classifier decisionTable

then mineAssociations
    from dataSet data1.dat
    present rules
    store in assoc1.skdql

then joinDisplay model1.skdql
            model2.skdql
            model3.skdql
            model4.skdql
            matId3.skdql
            assoc1.skdql
            store in result.skdql
```

O resultado final do processo, armazenado em *result.skdql*, é o seguinte:

```
Id3

areaMedium = CflMfr: PM_PlayOn
areaMedium = CblMfr: PM_PlayOn
areaMedium = CflMbl: PM_PlayOn
areaMedium = CbrMfl: PM_KickOff_Left
areaMedium = CflMfl: PM_PlayOn
areaMedium = CfrMfl: PM_PlayOn


J48 pruned tree
------------------
areaMedium = CflMfr: PM_PlayOn (6.0)
areaMedium = CblMfr: PM_PlayOn (8.0)
areaMedium = CflMbl: PM_PlayOn (2.0)
areaMedium = CbrMfl: PM_KickOff_Left (10.0/2.0)
areaMedium = CflMfl: PM_PlayOn (4.0)
areaMedium = CfrMfl: PM_PlayOn (2.0)

Number of Leaves  :     6
Size of the tree :     7
```

```
Id3

areaMedium = CflMfr: CflMfrFbl
areaMedium = CblMfr
|  number = 9: null
|  number = 2: CblMfrFbl
|  number = 10: null
|  number = 6: null
|  number = 7: CblMfrFfl
|  number = 11: null
areaMedium = CflMbl: CflMblFbr
areaMedium = CbrMfl: CbrMflFfl
areaMedium = CflMfl: CflMflFbl
areaMedium = CfrMfl: CfrMflFfl


Decision Table:

Number of training instances: 160
Number of Rules : 8
Non matches covered by Majority class.
Best first search for feature set,
terminated after 5 non improving subsets.
Evaluation (for feature selection): CV (leave one out)
Feature set: 2,3,5

Rules:
================================================
time            Y               player
================================================
'(124.9-142.6]' '(24.9-inf)'    10.0
'(54.1-71.8]'   '(24.9-inf)'    10.0
'(160.3-inf)'   '(15.6-18.7]'   9.0
'(142.6-160.3]' '(12.5-15.6]'   9.0
'(36.4-54.1]'   '(12.5-15.6]'   7.0
'(18.7-36.4]'   '(12.5-15.6]'   2.0
'(160.3-inf)'   '(-inf-0.1]'    11.0
'(-inf-18.7]'   '(-inf-0.1]'    8.4
================================================


Confusion Matrix - Id3:
  a  b  c  d  e  f  g  h   <-- classified as
  0  0  0  0  0  2  0  0 |  a = CflMfrFfl
  0  2  0  0  0  0  0  0 |  b = CblMfrFbl
  0  0  2  0  0  0  0  0 |  c = CflMblFbr
  0  0  0 10  0  0  0  0 |  d = CbrMflFfl
  0  0  0  0  6  0  0  0 |  e = CblMfrFfl
  0  0  0  0  0  4  0  0 |  f = CflMfrFbl
  0  0  0  0  0  0  4  0 |  g = CflMflFbl
  0  0  0  0  0  0  0  2 |  h = CfrMflFfl


Apriori
=======

Minimum support: 0.3
Minimum metric <confidence>: 0.9
Number of cycles performed: 14
```

```
Generated sets of large itemsets:

Size of set of large itemsets L(1): 5
Size of set of large itemsets L(2): 5
Size of set of large itemsets L(3): 1

Best rules found:

 1. description=PM_PlayOn 24 ==> periodFine=1 24    conf:(1)
 2. number=9 14 ==> periodFine=1 14    conf:(1)
 3. areaFine=CbrMflFfl 10 ==> periodFine=1 areaMedium=CbrMfl 10
    conf:(1)
 4. areaMedium=CbrMfl 10 ==> periodFine=1 areaFine=CbrMflFfl 10
    conf:(1)
 5. periodFine=1 areaFine=CbrMflFfl 10 ==> areaMedium=CbrMfl 10
    conf:(1)
 6. periodFine=1 areaMedium=CbrMfl 10 ==> areaFine=CbrMflFfl 10
    conf:(1)
 7. areaFine=CbrMflFfl areaMedium=CbrMfl 10 ==> periodFine=1 10
    conf:(1)
 8. areaFine=CbrMflFfl 10 ==> areaMedium=CbrMfl 10    conf:(1)
 9. areaMedium=CbrMfl 10 ==> areaFine=CbrMflFfl 10    conf:(1)
10. areaMedium=CbrMfl 10 ==> periodFine=1 10    conf:(1)
```

Avaliando os resultados apresentados, é verificado que:

- Segundo os classificadores Id3 e J48 - aplicados sobre dados filtrados (atributos relevantes) – cinco áreas possuem predominância do modo “jogo em andamento” (Play_On), a saber: CflMfr, CblMfr, CflMbl, CflMfl e CfrMfl;

- Quando o classificador Id3 é aplicado sobre dados não filtrados, ele fornece detalhamento em relação a subáreas (Coarse, Medium, Fine), cuja exata precisão pode ser conferida na matriz de confusão da tarefa. Neste caso, faz-se necessário observar o seguinte: o fato dos dados não terem sido filtrados em relação à relevância dos atributos permitiu que o algoritmo informasse a relação área/subárea (Coarse, Medium, Fine) - não se tratando portanto de uma informação nova, mas sim de uma definição de projeto refletida pelo algoritmo; decorrentemente, a classificação é “perfeita” gerando uma matriz de confusão com total “precisão”.

- Já a submissão de dados derivados de posse de bola (tposse.dat) a um algoritmo de tabela de decisão gera uma caracterização que relaciona o tempo do jogo, o eixo Y do posicionamento do jogador e a posse de bola deste.

- Na aplicação de associação ao conjunto de dados não filtrado (data1.dat), verifica-se que uma série de regras são geradas relacionando modo de jogo, jogador, área e

tempo de jogo. Neste caso, o algoritmo também é “enganado”, conforme pode ser verificado em algumas regras geradas.

Diante destas informações geradas pelos algoritmos, verifica-se que no caso da classificação com Id3 e J48, um algoritmo confirma o resultado do outro apresentando uma tendência de modo de jogo em áreas Medium do campo, apontando para uma atividade contínua (Play_On) nestas áreas. Constata-se também que a filtragem com relevância de atributos contribui para uma maior qualidade das informações, evitando armadilhas como no caso das áreas/subáreas. Outro ponto verificado é que nas demais tarefas não é gerado um padrão imediatamente compreensível, indicando assim que o conjunto de dados, sua formatação e/ou algoritmo de mineração devem ser modificados na busca por conhecimento neste domínio, num processo iterativo e interativo característico da descoberta de conhecimento em bancos de dados.

5.6 Teste da Implementação

Com o intuito de testar SKDQL e as funcionalidades que foram implementadas, uma bateria de tarefas de KDD utilizando a linguagem foi realizada sobre o banco de dados da RoboCup, um caso real de um domínio que abriga diferentes desafios com relação a KDD (dados esparsos, de baixa granularidade, com características espaço-temporais etc.). As tarefas apresentadas como exemplo no capítulo anterior (bem como o exemplo de uso supracitado) foram incluídas nos testes realizados.

Nas etapas de testes, diversos conjuntos de dados foram acessados e submetidos a manipulação, mineração e visualização de resultados através do protótipo de SKDQL. Uma vez que a implementação integra diferentes softwares, os testes revelaram inconsistências nesta integração. Entretanto, a análise sistemática seguida da modificação de configurações do software possibilitaram a eliminação de todos os problemas apresentados.

Nesta versão do protótipo não foi implementado nenhum recurso gráfico para a interface com o usuário, sendo esta totalmente textual, haja vista a sua suficiência para a apresentação e validação dos resultados gerados dentro da proposta de SKDQL (regras, árvores, estatísticas, matrizes etc.).

6 Conclusão

6.1 Contribuições da Dissertação

6.1.1 Uma Linguagem de Especificação para Descoberta de Conhecimento em Bancos de Dados

A proposta de SKDQL fornece, a priori, uma linguagem de especificação para a aplicação de diferentes tarefas de descoberta de conhecimento sobre bancos de dados, levando em conta características do processo, tais como:

- Iteratividade – peculiarmente, as tarefas de KDD demandam a aplicação e reaplicação de diferentes recursos e ferramentas para o bom andamento do trabalho em questão. SKDQL possibilita que acesso, pré-processamento, mineração de dados e manipulação do conhecimento a priori ou adquirido sejam aplicados quantas vezes forem necessárias dentro do ciclo de KDD (Figura 1).

- Interatividade – caracteristicamente, SKDQL possibilita que o analista efetue tarefas de forma contínua e interativa, solicitando tarefas e encadeando novas operações a resultados previamente alcançados, enquanto visualiza resultados intermediários.

- Operacionalização das tarefas de KDD – ao usuário são disponibilizados recursos num estilo muito semelhante a SQL, com primitivas específicas para cada tarefa, livrando o "minerador" (analista de dados que domina prioritariamente os conceitos e técnicas de KDD) de detalhes de implementação de ferramentas necessárias ao processo.

- Heterogeneidade – a atual proposta de SKDQL contempla o acesso a diferentes modelos de dados amplamente utilizados, aumentando a autonomia do usuário no que diz respeito ao tratamento de diferentes bases de dados dentro do mesmo ambiente.

- Integração – em função dos requisitos de KDD nos pontos supracitados, SKDQL integra recursos de diferentes ferramentas que usualmente são utilizados em ambientes distintos. Esta heterogeneidade lega ao usuário a solução de problemas mais relacionados às próprias bases de dados e à conciliação de ferramentas utilizadas do que ao processo de KDD em si. A proposta permite a utilização concomitante e uniforme de diferentes recursos num mesmo ambiente, além da possível incorporação de novas funcionalidades à linguagem de forma modular, através da agregação de componentes.

6.1.2 Implementação de uma Interface SKDQL para o Ambiente MATRIKS

Como parte integrante do MATRIKS, SKDQL fornece uma interface intuitiva, voltada ao usuário, que acessa de forma transparente outros componentes da arquitetura permitindo, inclusive, o incremento de funcionalidades através da integração de novas ferramentas, levando sempre em consideração a necessidade de abertura e modularidade do próprio MATRIKS.

Como linguagem declarativa e intuitiva de consulta, facilmente extensível, SKDQL propõe, de forma semelhante à contribuição de SQL a SGBD´s relacionais, uma padronização de linguagem para descoberta de conhecimento em bancos de dados, levando sempre em consideração a possibilidade de incorporação de aspectos e recursos relevantes ao processamento.

6.1.3 Criação de um Estudo de Caso de Descoberta de Conhecimento para o Projeto MATRIKS

O cubo de dados desenvolvido a partir dos logs da Robocup forneceu um estudo de caso muito rico para o problema em questão. Diferentes aspectos e problemas foram abordados, dentre eles:

- Obtenção dos dados – embora os logs dos campeonatos da RoboCup estivessem disponíveis para download, foi necessário tratá-los em diferentes fases, até que chegassem a um nível de granularidade e semântica aceitáveis.

Tal “exercício” de tratamento de dados demonstrou o quão valiosas são abordagens e ferramentas adequadas ao bom desempenho da tarefa.

- Granularidade – mesmo após diferentes fases de tratamento dos dados, observou-se que além de serem muito esparsos, sua granularidade também era muito baixa. Daí a necessidade de passos de derivação dos dados com o intuito de enriquecê-los, aumentando também sua granularidade.

- Domínio espaço-temporal – a dinâmica de um jogo de futebol realizado por simuladores envolve ações (chute, gol, passe etc.), em um momento determinado (cada jogo possui 2 tempos de 3000 ciclos cada), situado em um espaço pré-definido (o campo é mapeado em coordenadas X e Y).

- Modelagem – todas estas questões demandaram a análise de diferentes pespectivas, levando em conta a complexidade do domínio e a necessidade de um modelo apropriado às tarefas de KDD, resultando no trabalho apresentado na Figura 8.

- Ferramentas – o desenvolvimento de tarefas de mineração utilizando diferentes ferramentas para acesso, pré-processamento, mineração de dados e manipulação do conhecimento obtido revelou diferentes limitações das ferramentas utilizadas, além da eminente necessidade de um ambiente integrado que suporte o ciclo de trabalho de um processo de KDD. Algo que chama atenção no trabalho realizado é que o resultado de ciclos de tarefas de mineração podem ser exatamente a conclusão de que é necessário substituir ou agregar ferramentas para o progresso do trabalho em andamento. Esta observação valida a proposta de ambiente aberto e extensível do MATRIKS.

Diante deste contexto, a contribuição de SKDQL torna-se evidente na realização da tarefa de KDD apresentada na seção *Exemplo de Uso*. Através da linguagem, foi possível definir declarativamente de forma objetiva passos de seleção, preparação e mineração de dados, além de apresentação e armazenamento de resultados; o processo como um todo é totalmente administrado a partir de uma interface única.

Para efeitos de comparação, suponha-se esta mesma tarefa tendo de ser realizada diretamente através de interações com o SGBD, com o Prolog e com Weka. Facilmente

verifica-se que o usuário teria de dominar a operação de cada uma destas ferramentas, além de gerenciar as eminentes incompatibilidades entre seus modelo de dados, lançando mão de outros recursos para viabilizar seu trabalho (o que nem sempre seria possível naquele momento). Este tipo de experiência foi, de fato, vivenciado no estudo de caso para a descoberta de conhecimento em logs da RoboCup (anteriormente relatado).

Certamente, o tempo gasto na execução de um processo como este seria muito maior diante da necessidade de manipular e administrar o fluxo de dados tratados e informações geradas entre os diferentes ambientes. Conseqüentemente, o direcionamento das tarefas e a análise direta dos seus resultados seria bastante prejudicado nesta inquietação operacional, entre cliques, emissão de comandos e migração entre telas e ambientes, comprometendo a qualidade dos resultados gerados justamente pela ausência de uma interface que proporcionasse total controle sobre o processo como um todo.

6.2 Limitações e Trabalhos Futuros

6.2.1 Estender o Escopo de SKDQL

Devido à abrangência, evolução e dinâmica de KDD, verifica-se que a extensão do escopo de SKDQL é uma conseqüência natural da continuidade deste trabalho, uma vez que a atual especificação abrange recursos para um subconjunto dos possíveis passos num procedimento de descoberta de conhecimento. Além disso, novos métodos e algoritmos que venham a ser criados e que sejam inerentes ao domínio também são candidatos a esta extensão, seja para tarefas de pré-processamento, mineração de dados ou manipulação de conhecimento.

Além disso, outros modelos de dados devem ser contemplados no que diz respeito ao acesso a bases de dados. Bancos de dados orientados a objetos e modelos de dados de sistemas legados (Cobol [COU1999], por exemplo) são alguns dos casos que devem ser levados em consideração nos trabalhos futuros que envolvem SKDQL.

6.2.2 Estender o Escopo da Implementação

A atual implementação da linguagem contemplou parte dos métodos e algoritmos especificados para SKDQL. A extensão da implementação, através da incorporação dos demais recursos especificados, bem como a efetiva implantação de novas ferramentas

inerentes ao domínio inclusive com a integração dos demais componentes do ambiente MATRIKS, conferirão mais poder, abrangência e versatilidade ao software.

Um ponto fundamental neste contexto é a modularidade do MATRIKS, pois com sua arquitetura aberta e extensível seus componentes (inclusive SKDQL e suas ferramentas) poderão ser atualizados periodicamente, haja vista a evolução natural da maioria dos componentes integrados, cujos desenvolvedores constantemente aperfeiçoam, atualizam e disponibilizam versões de suas ferramentas.

6.2.3 Testar Especificação e Implementação com Maior Variedade de Tarefas de Descoberta de Conhecimento em Bancos de Dados

Embora uma série de testes tenham sido realizados, é fundamental que SKDQL ainda seja submetida a um conjunto mais amplo de tarefas e procedimentos com diferentes cenários, tanto no banco de dados da RoboCup como em outros domínios relevantes e complexos, com o intuito de aperfeiçoar os atuais recursos da linguagem e validar as funcionalidades já testadas.

Levando em conta a amplitude de tarefas, abordagens e ferramentas envolvidas nos processos de KDD, ressalta-se que SKDQL é apresentada como uma proposta alternativa para a padronização de linguagens de descoberta de conhecimento, não sendo considerada uma "solução" ou padrão a ser estabelecido, uma vez que a especificação e a implementação propõem-se a demonstrar a aplicação prática da linguagem num domínio real e complexo.

6.2.4 Integração com CWM

Da mesma forma que os repositórios de dados crescem desproporcionalmente à capacidade de análise dos mesmos através de métodos convencionais, verifica-se também uma diversificação cada vez maior dos padrões, formatos e tipos de dados destes imensos repositórios.

Neste contexto, as tarefas de KDD encontram um grande desafio no que diz respeito ao tratamento destes dados heterogenêos. Portanto, a integração de SKDQL com Common Warehouse Metamodel (CWM) apresenta-se como um ponto estratégico na viabilização de descoberta de conhecimento em diferentes fontes de dados, adotando assim um padrão que

em pouco tempo deve refletir a realidade de diferentes bases de dados, dentre as quais SGBD's, Data Warehouses e repositórios Web.

6.2.5 XKDQL: Versão XML de SKDQL

Extensible Markup Language (XML) [W3C2001] é um formato universal para estruturar documentos e dados na Web, facilitando o intercâmbio de ambos. A crescente adoção de XML para representação de dados, juntamente com a larga utilização da própria Web apontam para uma extensão da atual proposta: XKDQL, uma versão XML de SKDQL.

Diante dos vários recursos para acesso, processamento e formatação de dados, a "família XML" oferece uma série de características inerentes ao domínio de KDD, principalmente no que diz respeito à representação do conhecimento, independência de plataforma e ao amplo suporte a esta tecnologia. Não obstante, a oferta de dados nesta plataforma apresenta curva crescente, sendo um repositório abundante, relevante e distribuído (Web), com grande potencial para a descoberta de conhecimento.

Na atual versão de SKDQL, a representação de conhecimento, a especificação de métodos e o intercâmbio de dados atrelam-se a alguns padrões que, embora largamente adotados, apresentam limitações na expressividade de representação, na extensibilidade de funções e no suporte à troca de informações.

XKDQL teria então amplo suporte aos recursos disponibilizados pela tecnologia XML, superando uma série de dificuldades hoje enfrentadas pelas atuais ferramentas, especialmente em ambientes heterogêneos. Além disso, a Web Semântica [BER1999] certamente disponibilizará excelentes repositórios para as mais diferentes abordagens em descoberta de conhecimento na Internet.

Referências

[AGR1994] AGRAWAL, R.; SRIKANT, R. *Fast algorithms for mining association rules in large databases* . Proceedings of the International Conference on Very Large Databases, Santiago, Chile, 1994.

[ALA1999] ALAVI, M; LEIDNER, D. *Knowledge Management Systems: Issues, Challenges and Benefits.* Communications of the Association for Information Systems, volume 1, article 7, 1999.

[BER1999] BERNERS-LEE, T. et al. *The Semantic Web.* Disponível no site Scientific American (1999). URL: http://www.scientificamerican.com/2001/0501issue/0501berners-lee.html

[BOO1998] BOOCH, G. et al. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[BRA1999] BRATKO, I. *Prolog – Programming for Artificial Intelligence.* Addison-Wesley, 1999.

[BUE1998] BUENO, J. C. *KDCOM: A Knowledge Discovery Component Framework.* Artificial Intelligence Research Institute, Espanha, 1998.

[CHA1998] CHAUDHURI, S. *Data Mining and Database Systems: Where is the Intersection?* Microsoft Research, 1998.

[COR1997] CORNELL, G.; HORSTMANN, C. *Core Java.* Makron Books, 1997.

[COU1999] COUGHLAN, M. *Cobol Programming – Lectures, Tutorials, Examples.* Disponível no site de Michael Coughlan (1999). URL: http://www.csis.ul.ie/COBOL/

[CRI2000] CRISP-DM Consortium. *CRISP-DM – Cross Industry Standard Process for Data Mining.* Disponível no site CRISP-DM (2000). URL: http://www.crisp-dm.org/

[DBM2000] DBMINER TECHNOLOGY INC. *DBMiner Interprise 2.0* (2000). Disponível no site da DBMiner Technology. URL: http://www.dbminer.com/

[DEC2001] DECLARATIVA SOFTWARE. *InterProlog – A Prolog-Java Interface.* Disponível no site da Declarativa Software (2001). URL: http://www.declarativa.com/interprolog.htm

[DOR2000] DORER, K. *RoboCup Soccer Monitor*. Disponível no site de Klaus Dorer (2000). URL: http://cognition.iig.uni-freiburg.de/members/klaus/

[DST2001] DISTRIBUTED SYSTEMS TECHNOLOGY CENTRE. *A Textual Notation Generator for MOF Models*. Disponível no site da OMG (2001). URL: http://www.omg.org/news/meetings/uml/proposals/a_textual_notation_generator_for.htm

[FAV2000] FAVERO, E. *HYSSOP - Hypertext Summary System for Olap*. Tese de Doutorado. UFPE, 2000.

[FAY1993] FAYYAD, U. M.; IRANI, K. B. *Multi-interval Discretization of Continnuous-Valued for Classification Learning*. Proceedings of the 13th Int. Joint Conference on Artificial Intelligence, 1993.

[FAY1996] FAYYAD, U. M.; PIATESKY-SHAPIRO, G.; SMYTH, P. *From Data Mining to Knowledge Discovery: An Overview*. In: Advances in Knowledge Discovery and Data Mining, AAAI Press, 1996.

[FID2000] FIDALGO, R. N. *JODI*: *Uma API Java para Interoperabilidade entre Sistemas OLAP e Servidores OLE DB for OLAP*. Dissertação de Mestrado. UFPE, 2000.

[GOE1999] GOEBEL, M.; GRUENWALD, L. *A Survey of Data Mining and Knowledge Discovery Software Tools*. ACM SIGKDD, 1999.

[HAN1996] HAN, J. et al. DMQL: A Data Mining Query Language for Relational Databases. Simon Fraser University, 1996.

[HAN1998] HAN, J.; CHEE, S.; CHIANG, J. *Issues for On-Line Analytical Mining of Data Warehouses*. Simon Fraser University, 1998.

[HAN2001] HAN, J.; KAMBER, M. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2001.

[HIL2001] HILLMAN, C. *Entropy on the World Wide Web*. Disponível no site de Roland Gunesh (2001). URL: http://www.math.psu.edu/gunesch/entropy.html

[IBM2001] INTERNATIONAL BUSINESS MACHINES CO. *DB2 Intelligent Miner for Data* (2001). Disponível no site da IBM. URL: http://www-4.ibm.com/software/data/iminer/fordata/

[INM1996] INMON, W. *Building the Data Warehouse*. New York, Jonh Wiley & Sons, 1996.

[ISO1996] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION - ISO. *Syntatic Metalanguage – Extended BNF*. ISO/IEC 14977, 1996.

[KIM1996] KIMBALL, R. *The Data Warehouse Toolkit*. New York, Jonh Wiley & Sons, 1996.

[LIN2000] LINO, N. C. Q. *DOODCI*: *Uma API para Integração entre Bancos de Dados Multidimensionais e Sistemas Dedutivos*. Dissertação de Mestrado.UFPE, 2000.

[LIU1999] LIU, M. *Deductive Database Languages: Problems and Solutions.* ACM Computing Surveys, 1999.

[LIU1999a] LIU, M. *ROL2: Towards a Real Deductive Object-Oriented Database Languages.* University of Regina, Canada, 1999.

[LUD1999] LUDÄSCHER, B.; YANG, G.; KIFER, M. *FLORA: The Secret of Object-Oriented Logic Programming*. The Programmer's Reference, 1999.

[MCH2001] UNIVERSITY OF MICHIGAN. *Jvision XMI Generator.* Disponível no site da University of Michigan (2001). URL: http://intel.si.umich.edu/cfdocs/ si/expo/

[MCR2000] MICROTOOL BMGH. *ObjectIF – The XMI Generator.* Disponível no site da Microtool GmbH (2000). URL: http://www.microtool.de/objectiF/en/ sp_xmi.htm

[MEN2000] MENEZES, P.B. *Linguagens Formais e Autômatos.* Sagra Luzzato, 2000.

[MIC2000] MICROSOFT CORPORATION. *OLE DB for Data Mining.* Disponível no site da Microsoft Corporation (2000). URL: http://www.microsoft.com/data/oledb/dm.htm

[MIC2001] MICROSOFT CORPORATION. *OLE DB for OLAP*. Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/data/oledb/olap

[MIC2001a] MICROSOFT CORPORATION. *Microsoft ADO.* Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/data/ado/default.htm

[MIC2001b] MICROSOFT CORPORATION. *Microsoft COM.* Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/com

[MIC2001c] MICROSOFT CORPORATION. *Microsoft OLE DB.* Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/data/oledb/default.htm

[MIC2001d] MICROSOFT CORPORATION. *Microsoft ODBC.* Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/data/odbc/

[MIC2002] MICROSOFT CORPORATION. *Microsoft SQL Server*. Disponível no site da Microsoft Corporation (2002). URL: http://www.microsoft.com/sql/default.asp

[MIC2002a] MICROSOFT CORPORATION. *Microsoft Visio*. Disponível no site da Microsoft Corporation (2002). URL: http://www.microsoft.com/office/visio/default.htm

[MIC2002b] MICROSOFT CORPORATION. *Microsoft DCOM*. Disponível no site da Microsoft Corporation (2001). URL: http://www.microsoft.com/com/tech/dcom.asp

[OMG2001] OBJECT MANAGEMENT GROUP INC. *The Common Object Request Broker Architecture – CORBA*. Disponível no site da OMG (2001). URL: http://www.corba.org

[OMG2001a] OBJECT MANAGEMENT GROUP INC. *MOF – Meta-Object Facility*. Disponível no site da OMG (2001). URL: http://www.omg.org/technology/documents/formal/mof.htm

[OMG2002] OBJECT MANAGEMENT GROUP INC. *Site oficial da OMG – Object Management Group, (2002)*. URL: http://www.omg.org

[OMG2002a] OBJECT MANAGEMENT GROUP INC. *XMI - XML Metadata Interchange*. Disponível no site da OMG (1999). URL: http://www.omg.org/technology/xml/index.htm

[ORA2001] ORACLE CORPORATION. *Oracle Darwin*. Disponível no site da Oracle Corporation. (2001). URL: http://www.oracle.com/ip/analyze/warehouse/datamining/

[PET1999] PETERSON, T. *Microsoft OLAP Unleashed*. Sams Publishing, 1999.

[POO2002] POOLE, J. et al. *Common Warehouse Metamodel – An Introduction to the Standard for Data Warehouse Integration*. OMG Press, 2002.

[RES2000] RESEDAS TEAM. *Moderes Java*. Disponível no site da Loria (2000). URL: http://www.loria.fr/equipes/resedas/JSMAN/Moderes/

[ROB1997] ROBERTS, S. *An Introduction to Progol*. 1997.

[ROB2002] THE ROBOCUP FEDERATION. *The Robot World Cup Initiative (RoboCup)*. Disponível no site da RoboCup (2002). URL: http://www.robocup.org

[ROC2001] ROCHA, J.B. *Java Bridge to Prolog – JB2P (2001)*. Disponível no site: http://www.cin.ufpe.br/~jbrj/msc/courses/taias/jb2p

[ROC2001a] ROCHA, J.B. et al. *Gerador de Dados Derivados em Java e Prolog.* Trabalho em grupo da disciplina Tópicos Avançados em Inteligência Artificial Simbólica – Centro de Informática – UFPE, 2001.

[ROW2001] ROWAN UNIVERSITY. *The Wavelet Tutorial.* Disponível no site da Rowan University (2001). URL: http://engineering.rowan.edu/~polikar/WAVELETS/WTtutorial.html

[SAR1998] SARAWAGI, S.; AGRAWAL, R.; MEGIDDO, N. *Discovery-Driven Exploration of OLAP Data Cubes*. IBM Almaden Research Center, 1998.

[SIL1999] SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. *Sistema de Banco de Dados*. Makron Books, 1999.

[SPS2001] SPSS INC. *Clementine Data Mining Solution.* Disponível no site da SPSS Inc. (2001). URL: http://www.spss.com/clementine/

[STA2002] STATSOFT, INC. *Principal Components and Factor Analysis*. Disponível no site da StatSoft, Inc. (2002). URL: http://www.statsoftinc.com/textbook/stfacan.html

[STO2000] STONE, P. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer.* The MIT Press, 2000.

[SUN2001] SUN MICROSYSTEMS, Inc. *Java Developer Connection: Documentation and Training.* Disponível no site da Sun Microsystems (2001).URL: http://developer.java.sun.com/developer/infodocs/?frontpage-main

[SUN2001a] SUN MICROSYSTEMS INC. *Java Remote Method Invocation – Java RMI.* Disponível no site da Sun Microsystems (2001). URL: http://java.sun.com/products/jdk/rmi/

[SUN2002] SUN MICROSYSTEMS INC. *Java Database Connection - JDBC.* Disponível no site da Sun Microsystems (2002). URL: http://java.sun.com/products/jdbc

[W3C2001] WORLD WIDE WEB CONSORTIUM. *Extensible Markup Language – XML.* Disponível no site da W3C (2001). URL: http://www.w3.org

[W3C2002] WORLD WIDE WEB CONSORTIUM. *Site oficial do W3C – World Wide Web Consortium, (2002)*. URL: http://www.w3.org

[WAI2001] THE UNIVERSITY OF WAIKATO. *Weka 3 – Machine Learning Software in Java*. Disponível no site da University of Waikato (2001). URL: http://www.cs.waikato.ac.nz/ml/weka/index.html

[WAR1999] WARMER, J.; KLEPPE, A. *The Object Constraint Language.* Addison-Wesley, 1999.

[WEB2002] WEBGAIN INC. *Java Compiler Compiler – JavaCC.* Disponível no site da WebGain Inc. (2002). URL: http://www.webgain.com/products/java_cc/

[WIT2000] WITTEN, I.; FRANK, E. *Data Mining – Practical Machine Learning Tools.* Morgan Kaufmann, 2000.

[XSB2001] THE XSB RESEARCH GROUP. *XSB Prolog*. Disponível no site do XSB Research Group (2001). URL: http://xsb.sourceforge.net/