(Versão Revisada)

**O Processo de Extração de Conhecimento
de Base de Dados Apoiado
por Agentes de Software**

Robson Butaca Taborelli de Oliveira

Orientadora:

Profa. Dra. Solange Oliveira Rezende

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo como parte dos requisitos para a obtenção do título de Mestre na Área de Ciências de Computação e Matemática Computacional.

USP - São Carlos

Outubro/2000

Dedicatória

*Dedico este trabalho especialmente
aos meus pais Wilsan e Robustiano.*

Agradecimentos

Agradeço com muito amor aos meus pais, Wilsan e Robustiano, que me educaram e me nutriram com muita sabedoria e dignidade para enfrentar todos os possíveis desafios da vida. A conclusão vitoriosa deste trabalho é a prova do quão maravilhosos você minha Mãe e meu Pai são.

Um agradecimento em especial para minha orientadora, que sempre acreditou no meu potencial e me guiou para um trilha de grandes oportunidades na minha vida profissional. Solange, muito obrigado pela confiança e pela sua ajuda.

Agradeço as minhas irmãs Rúgina e Rúbia e a todos os parentes e amigos da minha cidade natal, Cuiabá, que sempre me apoiaram e incetivaram a minha busca por grandes desafios.

Agradeço aos meus amigos Roberto, Chandler e Walter, e amigas Jaqueline, Cristiane, Roberta e Tatiana que me socorreram nos momentos finais de entrega desta dissertação corrigindo, adicionando conceitos e sugerindo novas idéias para os desfecho desta dissertação.

Agradecimento aos velhos de guerra da festiva "Casa do Matite" Tatui, Tanão, Júlio, Taboca, Araxá, Dunga, Karim e Besta que me mostraram como se pode aprender muito e trocar idéias compartilhando a mesma casa. E é claro, agradeço por todas as vezes que vocês me acordaram (ou ao menos tentaram), pois talvez sem isso eu não teria finalizado ainda minha graduação.

Agradeço a todos meus amigos e amigas da Turma 93 que de alguma forma me incentivaram e me apoiaram para fazer o Mestrado.

Agradeço aos professores Maria Carolina e André do LABIC que me incentivaram na realização deste e de outros trabalhos.

Um agradecimento especial ao CNPq pelo auxílio financeiro que me foi dado nos meus períodos de pós-graduação e graduação.

Resumo

Os sistemas de aplicações científicas e comerciais geram, cada vez mais, imensas quantidades de dados os quais dificilmente podem ser analisados sem que sejam usadas técnicas e ferramentas adequadas de análise. Além disso, muitas destas aplicações são voltadas para Internet, ou seja, possuem seus dados distribuídos, o que dificulta ainda mais a realização de tarefas como a coleta de dados. A área de Extração de Conhecimento de Base de Dados diz respeito às técnicas e ferramentas usadas para descobrir automaticamente conhecimento embutido nos dados. Num ambiente de rede de computadores, é mais complicado realizar algumas das etapas do processo de KDD, como a coleta e processamento de dados. Dessa forma, pode ser feita a utilização de novas tecnologias na tentativa de auxiliar a execução do processo de descoberta de conhecimento. Os agentes de software são programas de computadores com propriedades, como, autonomia, reatividade e mobilidade, que podem ser utilizados para esta finalidade. Neste sentido, o objetivo deste trabalho é apresentar a proposta de um sistema multi-agente, chamado Minador, para auxiliar na execução e gerenciamento do processo de Extração de Conhecimento de Base de Dados.

Abstract

Nowadays, commercial and scientific application systems generate huge amounts of data that cannot be easily analyzed without the use of appropriate tools and techniques. A great number of these applications are also based on the Internet which makes it even more difficult to collect data, for instance. The field of Computer Science called Knowledge Discovery in Databases deals with issues of the use and creation of the tools and techniques that allow for the automatic discovery of knowledge from data. Applying these techniques in an Internet environment can be particularly difficult. Thus, new techniques need to be used in order to aid the knowledge discovery process. Software agents are computer programs with properties such as autonomy, reactivity and mobility that can be used in this way. In this context, this work has the main goal of presenting the proposal of a multiagent system, called Minador, aimed at supporting the execution and management of the Knowledge Discovery in Databases process.

Sumário

1	Introdução	1
1.1	Contexto	1
1.2	Motivação e Relevância	2
1.3	Objetivos	3
1.4	Organização	3
2	Extração de Conhecimento de Base de Dados	5
2.1	Definição	6
2.2	Armazenamento de Dados	8
2.2.1	<i>Data Warehouse</i>	8
2.2.2	Base de Dados x <i>Data Warehouse</i>	10
2.3	Processo de Extração de Conhecimento de Base de Dados	12
2.3.1	Compreensão do Domínio de Aplicação	14
2.3.2	Seleção de Dados	15
2.3.3	Preparação de Dados	16
2.3.4	Limpeza e Pré-processamento de Dados	17
2.3.5	<i>Data Mining</i>	18
2.3.5.1	Tarefas de <i>Data Mining</i>	19
2.3.5.2	Algoritmos Usados em <i>Data Mining</i>	20
2.3.5.3	Extração de Padrões	23
2.3.6	Interpretação e Avaliação de Padrões	24
2.3.7	Consolidação do Conhecimento	25
2.4	Áreas de Apoio	25
2.4.1	<i>Data Warehousing</i>	26

2.4.2	Estatística	26
2.4.3	Visualização de Dados	27
2.5	Alguns Desafios da Área de Extração de Conhecimento de Base de Dados . .	27
2.6	Considerações Finais	28
3	Agentes de Software	30
3.1	Definição	31
3.2	Tipologia de Agentes e de Ambientes	34
3.2.1	Tipos de Agentes de Software	34
3.2.2	Tipos de Ambientes	37
3.3	Projeto de Agente de Software	38
3.3.1	Arquiteturas de Agentes	38
3.3.2	Linguagens de Agentes	38
3.4	Comunicação Entre Agentes	39
3.4.1	KIF	40
3.4.2	Ontologias	40
3.4.3	KQML	41
3.5	Desafios para a Construção de Agentes	42
3.6	Considerações Finais	43
4	Proposta do Sistema Minador	45
4.1	Motivação	46
4.2	KDD Apoiado por Agentes de Software	48
4.2.1	Little-JIL	49
4.2.2	InfoSleuth	50
4.2.3	Considerações sobre os Projetos InfoSleuth e Little-JIL	51
4.3	Visão Geral do Sistema Minador	51
4.3.1	Semelhanças entre Trabalhos Relacionados	52
4.3.2	Diferenças entre Trabalhos Relacionados	53
4.3.3	Contribuições Oferecidas	53
4.4	Arquitetura do Minador	55
4.5	Interface de <i>Web</i>	56
4.5.1	Interface Gráfica do Usuário	57

4.5.2	Funcionamento da Interface	58
4.6	Gerenciador de Descobrimento	58
4.6.1	Módulo de Configuração de Instâncias	59
4.6.2	Módulo Monitor do Sistema	60
4.6.3	Módulo de Processamento de Dados	60
4.6.4	Módulo de Mineração	61
4.6.5	Módulo de Distribuição	61
4.7	Agentes	62
4.7.1	Agente Base	63
4.7.1.1	Reatividade	63
4.7.1.2	Pró-Atividade	65
4.7.1.3	Mobilidade	66
4.7.1.4	Persistência	69
4.7.1.5	Máquina de Estados Finitos	70
4.7.1.6	Plano de Trabalho	71
4.7.1.7	Plano de Ação	72
4.7.1.8	Canais de Comunicação	72
4.7.1.9	Memória de Trabalho	73
4.7.1.10	Identificação Única	74
4.7.1.11	Política de Segurança	74
4.7.2	Agente de Dados	75
4.7.2.1	Agente Coletor	77
4.7.2.2	Agente de Arquivo	78
4.7.2.3	Agente de Internet	78
4.7.2.4	Agente Processador	79
4.7.3	Agente Minerador	80
4.7.4	Agente Distribuidor	83
4.8	Ambiente de Execução de Agentes	84
4.9	Consideração Sobre Implementações e Testes Realizados	85
4.9.1	Peculiaridades da Implementação	86
4.9.2	AEA	87
4.9.3	Coleta Automática de Dados e Tratamento dos dados	87

4.10 Protótipo Elaborado	88
4.11 Considerações Finais	92
5 Conclusões e Propostas para Trabalhos Futuros	94
Bibliografia	98

Lista de Figuras

2.1	Visão geral das etapas do processo de KDD.	12
4.1	Arquitetura do Sistema Minador.	55
4.2	Interface de <i>Web</i> do Sistema Minador.	57
4.3	Organização dos Módulos do Gerenciador de Descobrimento.	59
4.4	Classes básicas de agentes do Minador.	62
4.5	Um evento sendo disparado para um agente base.	64
4.6	Ilustração do esquema de eventos de um agente base.	65
4.7	Agente situado em um AEA.	67
4.8	Agente transferindo-se de um AEA a outro.	68
4.9	Máquina de Estados Finitos dos agentes do sistema.	71
4.10	Canais de Comunicação de um agente.	73
4.11	Agentes de Dados do sistema Minador.	76
4.12	Agente Minerador do sistema Minador.	81
4.13	Agente Distribuidor do sistema Minador.	83
4.14	Ambientes de Execução de Agentes.	85
4.15	Modelo físico da base de dados do Minador.	91

Lista de Tabelas

2.1	Principais diferenças entre Base de Dados Operacionais e <i>Data Warehouse</i> . .	11
2.2	Classificação dos algoritmos de AM usados em <i>Data Mining</i>	22
2.3	Exemplos de algoritmos de Aprendizado de Máquina.	23
4.1	Tabela de Decisão	66
4.2	Exemplo de um itinerário a ser seguido por um agente.	68
4.3	Histórico de vida de um agente	69
4.4	Viagens feitas por um agente	69
4.5	Base de Relacionamentos de um agente	70
4.6	Plano de Trabalho de um agente	71
4.7	Plano de Ação de um agente	72
4.8	Política de Segurança	74
4.9	Comandos da <i>Structured Query Language</i>	75
4.10	Plano de Coleta de um agente coletor.	77
4.11	Cláusulas principais da sentença SELECT do SQL.	77
4.12	Plano de Coleta de um agente de arquivos	78
4.13	Exemplo de um arquivo de configuração utilizado pela ferramenta MineSet. .	81
4.14	Exemplo de um arquivo de dados utilizado pela ferramenta MineSet.	81
4.15	Implementação do Protótipo de Interface de <i>Web</i>	88
4.16	Implementação do Protótipo do Gerenciador de Descobrimento.	89
4.17	Implementação dos Agentes do Protótipo.	90
4.18	Implementação dos AEA.	90

Capítulo 1

Introdução

1.1 Contexto

Os avanços conseguidos nas áreas de hardware e software possibilitaram a criação de aplicações científicas e comerciais capazes de processar imensos volumes de dados. Por exemplo, o sistema que é usado pela empresa *Wal-Mart* para computar as vendas da empresa processa mais de 20 milhões de transações diariamente produzindo Gigabytes (10^9 bytes) de dados. Ainda, os sistemas criados para aplicações científicas como o *Earth Observing System* (EOS) da NASA que coleta Gigabytes de dados por hora contendo informações sobre a atmosfera, oceanos e florestas da Terra (Silberschatz et al., 1995).

Cada vez mais, as organizações vêm fazendo vultosos investimentos em equipamentos e aplicativos usados para aperfeiçoar o armazenamento, gerenciamento, integração e entendimento dos seus dados. Isto se deve a uma mudança de filosofia, pois, hoje em dia, as bases de dados não podem mais ser consideradas simples repositórios de informações, mas sim, um importante patrimônio da organização (Sen and Jacob, 1998).

A quantidade de dados gerada pelos sistemas das organizações, supera a capacidade humana de interpretar e compreender tanta informação. Por isso, são necessárias novas técnicas e ferramentas capazes de analisar automaticamente a imensidão de dados produzidos, fornecendo o conhecimento necessário para ajudar nos processos de tomada de decisão.

A área da Computação, conhecida por Extração de Conhecimento de Base de Dados ou *Knowledge Discovery in Databases* (KDD), surgiu para auxiliar a análise de grande volume de dados. As pesquisas nesta área têm o objetivo de estudar a aplicação de no-

vas metodologias, técnicas e ferramentas capazes de extrair conhecimento embutido em um grande volume de dados.

O processo de KDD é feito utilizando-se conceitos de Bases de Dados, métodos estatísticos, ferramentas de visualização e técnicas de Inteligência Artificial (IA). Assim, por ser uma ciência multidisciplinar, as inovações feitas nestas áreas são refletidas diretamente nas metodologias, técnicas e ferramentas usadas para a descoberta de conhecimento nos dados.

1.2 Motivação e Relevância

O uso da tecnologia de informática vem, há alguns anos, sofrendo diversas mudanças, principalmente, no que tange o acesso e análise de dados. Estas transformações estão moldando um novo paradigma baseado no acesso a *Web*, na interoperabilidade de aplicações, na independência de local físico e na disponibilização contínua de informações de todos os tipos e formatos através da Internet. Conseqüentemente, as empresas estão começando a explorar as possibilidades oferecidas pela Internet para mudar o seu processo e modelo de negócios.

As implicações destas mudanças para o mundo dos negócios são enormes. Assim, este novo paradigma envolve o uso de sistemas especificamente projetados para comunicar com o usuário ou com outros sistemas através da Internet de forma flexível para entregar informações úteis, universalmente acessíveis, disponíveis continuamente e em tempo hábil para análise e atualizações (Olofson, 1998).

É de interesse das organizações aplicar algumas das técnicas de KDD na tentativa de se encontrar conhecimento nesta nova realidade na qual a Internet figura como componente central. Todavia, é preciso fazer adaptações das técnicas, metodologias e ferramentas usadas na extração de conhecimento de bases de dados. Dessa forma, como o ambiente de transações encontra-se distribuído pela Internet a complexidade para se recuperar e processar os dados é maior. Ainda, o conhecimento obtido precisa chegar às pessoas envolvidas no processo de descoberta, independente do local físico, para que seja possível avaliar e utilizar tudo aquilo que foi encontrado. Portanto, é interessante buscar tecnologias capazes de apoiar a execução do processo de Extração de Conhecimento de grande volumes de dados.

Recentemente, os chamados agentes de software têm sido foco de intensas pesquisas na comunidade científica. A utilização dos agentes é bastante variada, eles estão sendo u-

sados, por exemplo, para buscar e gerenciar informações na Internet, viabilizar o comércio eletrônico, em aplicações médicas e entretenimento. Os agentes podem possuir propriedades como, autonomia, reatividade e mobilidade, que permitem ao agente, respectivamente, executar tarefas sem a intervenção do usuário, reagir a eventos ocorridos e viajar por entre uma rede de computadores (Bradshaw, 1997a).

Uma possível abordagem capaz de viabilizar a extração de conhecimento, no ambiente da Internet, seria a utilização de agentes de software. Estes agentes podem ajudar os usuários na realização de tarefas simples como a coleta de dados. Algumas tarefas repetitivas podem ser executadas pelos agentes, por exemplo, a transformação nos dados. Isto pode permitir que o usuário concentre-se mais em outras etapas do KDD. A distribuição do conhecimento obtido também pode ser feita através destes agentes.

1.3 Objetivos

Visando contribuir com uma solução capaz de auxiliar a realização semi-automática do processo de KDD num ambiente de computação em rede, foi realizado um estudo referente aos principais conceitos das áreas de Extração de Conhecimento de Base de Dados e de Agentes de Software. Assim, o objetivo geral do projeto é a proposição de um sistema multi-agente para apoiar a execução do Processo de Extração de Conhecimento de Bases de Dados num ambiente de rede de computadores (e.g. Internet).

Uma vez elaborada idéia do sistema, alguns “sub-objetivos” poderão ser alcançados:

- Verificação da viabilidade de se construir um sistema deste porte
- Sugestão de alternativas às soluções atualmente propostas
- Auxiliar na implementação de sistemas menos complexo através da documentação da proposta

1.4 Organização

Neste capítulo foi apresentado o contexto geral do trabalho, uma motivação para o desenvolvimento da pesquisa, bem como os objetivos pretendidos. O restante da dissertação encontra-se logicamente dividida em duas partes. Na primeira é apresentada uma visão geral

das áreas de Extração de Conhecimento de Base de Dados e Agentes de Software. Na segunda parte, é apresentado a proposta do sistema Minador e algumas considerações finais e trabalhos futuros.

Esta dissertação possui cinco capítulos organizados da seguinte forma: o capítulo atual, Capítulo 1, apresenta a introdução dos assuntos tratados no trabalho. No Capítulo 2 está apresentada uma revisão dos aspectos principais da área de Extração de Conhecimento de Base de Dados. O capítulo seguinte, Capítulo 3, apresenta uma revisão dos conceitos existentes na área de Agentes de Software. O Capítulo 4 é o capítulo central da dissertação, onde está apresentada a proposta do Sistema Minador. Finalmente, o Capítulo 5 apresenta as considerações finais e algumas propostas de trabalhos futuros.

Capítulo 2

Extração de Conhecimento de Base de Dados

A evolução da computação ocorrida nos últimos anos possibilitou um aumento na capacidade de processamento e armazenamento de dados. A facilidade atual que uma aplicação, seja científica ou comercial, possui para gerar *gigabytes* ou *terabytes* de dados em poucas horas excede em muito a capacidade de pesquisadores e analistas de mercado em fazer análises sobre os mesmos.

Os aplicativos que, normalmente, são utilizados para consultas (e.g. planilhas eletrônicas) foram projetados para gerar relatórios simplificados de atividades como: listar o número de itens vendidos no dia, verificar estoques de vendas, desenhar gráficos da evolução das vendas obtidas, dentre outros. Esses tipos de consultas eram capazes de satisfazer às necessidades rotineiras de uma empresa.

Atualmente, os analistas de negócios precisam fazer uso de ferramentas capazes de responder perguntas mais complexas como: “qual produto de alta lucratividade venderia mais com a promoção de um item de baixa lucratividade, analisando-se os dados dos últimos 10 anos de vendas?”. Este tipo de informações pode ser crucial para a sobrevivência de uma empresa nos tempos de hoje. Dessa forma, novas ferramentas de análises e extração de conhecimento devem ser usadas no processo decisório.

A realização de *Data Warehousing* (Gardner, 1998) é considerado como um dos primeiros passos para se tornar factível a análise em grande quantidade de dados no apoio ao processo decisório. Essa abordagem tem como objetivo básico criar um repositório, conheci-

do por *Data Warehouse*, que contenha dados limpos, agregados e consolidados que possam ser analisados por ferramentas OLAP (*On-Line Analytical Processing*). Essas ferramentas apresentam facilidades para a realização de consultas complexas em bases de dados multidimensionais.

As ferramentas utilizadas para analisar um *Data Warehouse*, normalmente, são orientadas às consultas, ou seja, são dirigidas pelos usuários, os quais possuem hipóteses que gostariam de comprovar, ou simplesmente, executam consultas aleatórias. Essa abordagem dependente do usuário pode impedir que padrões escondidos nos dados sejam encontrados de forma “inteligente”, uma vez que o usuário não terá condições de imaginar todas as possíveis relações e associações existentes em um grande volume de dados. Por isso, faz-se necessário a utilização de técnicas de análise dirigidas por computador que possibilitem a extração automática (ou semi-automática) de novos conhecimentos de um grande repositório de dados (Bradley et al., 1998).

O interesse em automatizar o processo de análise de grande quantidade de dados tem estimulado várias pesquisas em um campo emergente chamado Extração de Conhecimento de Bases de Dados (*KDD - Knowledge Discovery in Database*). Os participantes dessa nova área de atuação visam elaborar técnicas, metodologias e ferramentas para apoiar a descoberta de conhecimento inédito e aumentar o grau de compreensão de um domínio de aplicação específico (Fayyad et al., 1996c).

A Extração de Conhecimento de Base de Dados é uma área multidisciplinar que incorpora técnicas utilizadas em diversas áreas como Base de Dados, Inteligência Artificial e Estatística. Desse modo, as técnicas utilizadas em KDD não devem ser vistas como substitutas de outras formas de análises (e.g. OLAP), mas sim, como práticas para melhorar os resultados das explorações feitas com as ferramentas atualmente usadas (Fayyad, 1996).

2.1 Definição

A definição aceita por diversos pesquisadores de KDD foi dada por Fayyad (Fayyad et al., 1996b) como sendo: “Extração de Conhecimento de Base de Dados é o processo não trivial de identificação de padrões válidos, novos, potencialmente úteis e compreensíveis embutidos nos dados”. Para compreender melhor o conteúdo desta esparsa definição, deve-se olhar individualmente cada componente da mesma:

- *Dados*: conjunto de fatos ou casos em um repositório de dados. Por exemplo, os valores dos campos de um registro de vendas em uma base de dados qualquer;
- *Padrão*: denota os conjuntos de modelos que representam alguma abstração de um subconjunto dos dados em alguma linguagem descritiva;
- *Processo*: a extração de conhecimento de Base de Dados envolve diversas etapas como a preparação dos dados, busca por padrões e avaliação do conhecimento;
- *Válidos*: os padrões descobertos devem possuir algum grau de certeza. Ou seja, devem satisfazer funções ou limiares que garantam que os exemplos cobertos e os casos relacionados ao padrão encontrado sejam aceitáveis;
- *Novos*: um padrão encontrado deve fornecer novas informações sobre os dados. O grau de novidade serve para determinar o quão novo ou inédito é um padrão. Ele pode ser medido através de comparações entre as mudanças ocorridas nos dados ou no conhecimento anterior;
- *Úteis*: os padrões descobertos devem ser incorporados de forma a serem utilizados;
- *Compreensíveis*: um dos objetivos de realizar-se KDD é fazer com que os padrões possam ser entendidos em alguma linguagem descritiva pelos usuários de forma a permitir uma compreensão mais profunda dos dados;
- *Conhecimento*: o conhecimento é definido em termos dependentes do domínio que estão relacionados fortemente com medidas de utilidade, originalidade e compreensão.

Vale ressaltar que vários autores usam o termo *Data Mining* para referenciar ao processo de Extração de Conhecimento de Base de Dados, porém, neste trabalho os termos KDD e *Data Mining* são usados com significados distintos. KDD refere-se ao processo como um todo, enquanto que *Data Mining* (como está exposto nas próximas seções) é tratado como um componente dentro deste processo.

A matéria-prima fundamental do KDD é composta pelos dados, que podem estar contidos em diferentes estruturas de armazenamento (e.g. banco de dados). A próxima seção aborda com mais detalhes algumas das estruturas freqüentemente utilizadas para a Extração de Conhecimento de Base de Dados.

2.2 Armazenamento de Dados

Todo o processo de KDD é orientado em função de seu domínio de aplicação e dos repositórios de dados inerentes ao mesmo. Para se fazer uso destes dados é necessário que eles estejam estruturados de forma a serem consultados e analisados de maneira adequada.

Os sistemas de aplicações, conhecidos por OLTP (*On-Line Analytical Processing*), processam dados armazenados em base de dados relacionais usadas para armazenar, consultar e alterar informações. Normalmente, não é possível aplicar as técnicas de KDD diretamente a estas bases, pois isso poderia resultar numa sobrecarga de consultas nas mesmas podendo literalmente “travar” um sistema, impossibilitando qualquer outro tipo de operação transacional. Assim, é recomendável que os dados a serem utilizados na descoberta de conhecimento estejam separados da base de dados operacional.

Uma solução para armazenar grandes quantidades de dados históricos de uma organização é a utilização de uma estrutura de armazenamento que viabiliza acessos otimizados aos dados previamente consolidados de uma organização, conhecidos por *Data Warehouse* (DW), reduzindo consideravelmente o tempo de realização processo de KDD. Uma visão geral sobre esta estrutura especializada de armazenamento e consulta é apresentada nas subseções seguintes.

2.2.1 *Data Warehouse*

Um *Data Warehouse* é uma coleção de dados orientada a fatos, multidimensional, integrada e não volátil que armazena uma “cópia” otimizada dos dados operacionais para serem consultados e analisados. Esse tipo de repositório possui dados agregados, resumidos e históricos para apoiar tomadas de decisões estratégicas. Um DW pode ser visto como uma evolução dos sistemas gerenciadores de informação (Hammer et al., 1995; Wu and Buchmann, 1997; Samos et al., 1998).

Existem alguns pontos importantes que precisam ser considerados no momento de se planejar e projetar um *Data Warehouse*, como: a eficiência nas repostas de consultas complexas, facilidade para se aumentar e diminuir o nível de agregação durante a navegação entre os dados, dispor de estruturas que possuam dados resumidos e agregados em múltiplas dimensões, entre outras.

Um projeto de *Data Warehouse* envolve a definição de um modelo estrutural, a de-

terminação de sua topologia e possíveis fontes de dados e os mecanismos para transformar, inserir e atualizar dados. Ainda, existem vários componentes auxiliares que viabilizam a construção e utilização do DW. Para se conseguir realizar um projeto que atenda aos requisitos de um determinado domínio de aplicação, basicamente, deve-se realizar:

- Escolha da fonte de dados: os dados que constituem um DW podem vir principalmente de bases de dados operacionais que armazenam transações continuamente processadas em uma organização. E, como uma empresa pode utilizar várias bases de diferentes vendedores, deve existir mecanismos para se fazer à integração das mesmas, viabilizando, assim, o armazenamento. Ainda, pode-se retirar dados de fontes externas como documentação existente, dados multimídia e Internet;
- Definição do modelo estrutural: todos os modelos seguem, basicamente, uma arquitetura formada por tabelas fatos, que possuem uma chave composta constituída de várias chaves que referenciam outras tabelas conhecidas por tabelas dimensionais, as quais representam diferentes dimensões de um negócio, como, por exemplo, cliente, vendas e produtos. Os modelos usados são o *Star* que possui uma única tabela fato ligada às várias tabelas dimensionais, o *Join Star* que possui diversas tabelas fatos ligadas às tabelas dimensionais, sendo que uma tabela dimensional pode ser referenciada por mais de uma tabela fato, e finalmente, o *Snowflake* que se diferencia do *Star* por possuir “subtabelas” dimensionais representando uma agregação maior dos dados que são referenciadas unicamente pelas tabelas dimensionais, por exemplo, a tabela Vendas possui uma chave ligando à tabela Lojas;
- Topologias: um DW possui topologias que o caracteriza como sendo centralizado, *Data Mart* ou distribuído. Um DW centralizado contém dados devidamente condicionados em fatos de interesses para usuários de unidades de múltiplos negócios da empresa. Um *Data Mart* contém um escopo menor dos dados, normalmente caracterizado por um único assunto ou única função de negócio. E um *Data Warehouse* distribuído consiste de vários DW conectados via rede com grande suporte ao processamento distribuído, permitindo que usuários em diferentes regiões geográficas conectados à qualquer DW possam trabalhar como se os dados residissem em um *Data Warehouse* único e centralizado, apesar dos mesmos estarem fisicamente distribuídos em múltiplos DW;

- Transformação dos dados: diversas operações de transformação podem ser feitas, como, a integração de dados vindos de diferentes fontes, limpeza, agregação e filtragem. Novas estruturas para armazenar dados podem ser definidas, a utilização de índices e *caches* conseguem otimizar as respostas de consultas;
- Atualizações: à medida que novos dados relevantes tornam-se disponíveis ou são modificados, os mesmos devem ser extraídos de suas fontes, traduzidos em um modelo comum e integrados aos dados existentes. Pode-se fazer uso de técnicas para detectar mudanças como, *triggers* nas fontes de dados que avisam ao gerenciador do DW e comparação de tabelas.

Os componentes que podem ser usados para auxiliar a construção de um *Data Warehouse*, são:

- Fontes de Dados: locais de onde são retirados os dados “crus” para serem processados;
- Integrador: ferramenta que assimila os dados de diferentes fontes reunindo-os;
- Transformadores: ferramentas que fazem a transformação dos dados retirados das fontes de dados e integrados;
- *Operational Data Store*: estrutura auxiliar que mantém os dados para propósitos de verificação e validação, sendo que uma vez consolidados, os mesmos são transferidos para um *Data Warehouse*;
- Interface Gráfica: ferramentas utilizadas para permitir a fácil navegação, visualização e consultas aos dados.

2.2.2 Base de Dados x *Data Warehouse*

Tipicamente, um DW é mantido separado da base de dados operacional. Isto é feito porque tal repositório suporta aplicações de processamento analítico “on-line” (OLAP), cujos requisitos funcionais e de desempenho são muito diferentes dos de aplicações para processamento de transações “on-line” (OLTP) que são suportadas por base de dados operacionais.

Os sistemas OLTP são utilizados para processar operações feitas diariamente em uma organização, por exemplo, um sistema de transações bancárias. Dessa forma, é necessário

que as estruturas de armazenamento desses sistemas possuam um modelo normalizado, como Entidade-Relacionamento, que favoreça o desempenho das transações e a consistência dos dados. Essas bases, normalmente, possuem entre *megabytes* a *gigabytes* de dados armazenados.

Num sistema OLAP, o *Data Warehouse* é projetado para apoiar consultas complexas que auxiliem os analistas durante o processo de tomada de decisão, assim, dados históricos, agregados e resumidos são necessários. Os dados consolidados de um DW podem ser gerados a partir de diversas fontes, logo este tipo de repositório tende a possuir *gigabytes* ou *terabytes* de dados (Chaudhuri and Dayal, 1997). A tabela 2.1 apresenta um resumo das principais diferenças entre base de dados operacionais e DW (Wu and Buchmann, 1997; Bontempo and Zagelow, 1998).

Aspectos	Base de Dados Operacional	Data Warehouse
Usuários	Projetista de Sistema Administrador de Sistema Operadores	Analistas Usuários do Conhecimento Executivos
Função	OLTP	OLAP
Projeto	Orientado a Aplicação	Orientado a Domínio
Modelo	Normalizado	Desnormalizado
Dados	Atuais, Relacionais Isolados	Históricos, Resumidos Integrado
Uso	Repetitivo	Ad-hoc
Acesso	Escrita/Leitura Transações simples	Leitura Consultas complexas
Requisitos	Desempenho de transações Consistência de Dados	Desempenho de Consultas Precisão nos Dados

Tabela 2.1: Principais diferenças entre Base de Dados Operacionais e *Data Warehouse*.

Apesar de não ser obrigatória, a construção de um *Data Warehouse* pode reduzir drasticamente a complexidade e a duração do processo de KDD. Na próxima seção, é apresentada uma visão geral do processo de Extração de Conhecimento de Base de Dados.

2.3 Processo de Extração de Conhecimento de Base de Dados

Extrair conhecimento de base de dados pode envolver, entre outras coisas, a utilização de algoritmos de Aprendizado de Máquina capazes de generalizar os fatos encontrados em um grande repositório de dados na forma de regras de alto nível compreensíveis ao ser humano.

As técnicas de Aprendizado de Máquina parecem estimular muito os pesquisadores da área e talvez seja a parte do processo de KDD que contenha substancialmente os maiores conteúdos científicos estudados. Todavia, quando se tenta realizar tarefas de descobrimento de conhecimento em aplicações do mundo real, percebe-se que as mesmas podem ser extremamente complexas e que a tarefa de *Data Mining* representa apenas uma porção (importante) do processo global (Brachman and Anand, 1996). Assim, o processo de KDD deve ser visto como sendo composto por várias etapas interligadas.

Figura 2.1: Visão geral das etapas do processo de KDD.

O processo de Extração de Conhecimento de Base de Dados é interativo e iterativo, envolvendo diversas etapas, como mostra a Figura 2.1. Deve-se considerar todos os seguintes passos para se conseguir extrair o conhecimento (Fayyad et al., 1996b):

1. Compreensão do domínio de aplicação;
2. Seleção de dados;
3. Preparação dos dados;
4. Limpeza dos dados;
5. Data Mining;
6. Interpretação e Avaliação do conhecimento;
7. Consolidação do conhecimento.

As etapas do processo de KDD, normalmente, são realizadas de forma sequencial, ou seja, é preciso compreender o domínio de aplicação, selecionar e transformar os dados para depois tentar encontrar padrões nos dados. E por se tratar de um processo iterativo, as pessoas envolvidas na sua realização devem possuir um canal de comunicação que viabilize uma boa troca de informações. Estes usuários possuem diferentes especialidades e necessidades, e cada um possui o seu papel dentro do processo. As classes dos usuários envolvidos no processo de KDD são:

- *Especialista do Domínio*: possui amplo entendimento do domínio da aplicação. Por exemplo, durante a análise de uma coleção de dados sobre vendas de uma empresa, o especialista pode ser um diretor de marketing.
- *Analista*: executa o processo de KDD e, portanto, deve possuir amplo conhecimento das etapas que compõem esse processo. Por exemplo, os analistas de sistemas, estatísticos e administradores de base de dados.
- *Usuário Final*: o qual pode não possuir amplo conhecimento do domínio. É frequentemente aquele que utiliza o conhecimento extraído no processo de KDD para auxiliá-lo em um processo de tomada de decisão.

Durante a realização do processo de KDD, a interação e comunicação entre o analista, especialista do domínio e usuário final é muito importante, uma vez que o sucesso da extração de conhecimento depende da comunicação entre eles (Brachman and Anand, 1996). Nesse

ínterim, é possível melhorar a troca de informações entre os envolvidos no processo utilizando-se técnicas de visualização de dados adequadas (Figura 2.1) para auxiliar na execução das etapas do processo (Rezende et al., 1998b).

As etapas do processo de KDD sugeridas podem não cobrir exaustivamente todos passos existentes, entretanto, esta seqüência de tarefas pode ser considerada como sendo os requisitos mínimos para se conseguir extrair conhecimento de bases de dados. Todas estas etapas são estudadas e detalhadas nas próximas seções.

O entendimento dos passos necessários para realizar o processo de KDD é de grande importância para este trabalho, assim, um detalhamento mais aprofundado sobre cada etapa faz-se necessária. As próximas seções, tratam dos passos considerados essenciais para a extração de conhecimento de grande quantidade de dados.

2.3.1 Compreensão do Domínio de Aplicação

Um aspecto importante no processo de KDD é o entendimento do domínio de aplicação, de onde será extraído o conhecimento necessário para apoiar a tomada de decisões. Para isso, fica evidente a necessidade de identificar quais são os objetivos e requisitos necessários para que seja possível extrair e incorporar o conhecimento conseguido.

Não existem perguntas padrões cujas respostas possam delinear um domínio de aplicação, uma vez que cada situação mantém suas peculiaridades próprias. Porém, alguns aspectos podem ser levados em consideração na tentativa de se definir a esfera de ação de uma aplicação, como:

- Definição dos objetivos e dos requisitos da extração de conhecimento;
- Identificação das possíveis fontes de dados;
- Pesquisa do conhecimento prévio existente;
- Modo de distribuição do conhecimento extraído;
- Incorporação do conhecimento extraído num sistema de tomada de decisão;
- Estudo de viabilidade e custos da aplicação do processo.

Uma vez que o domínio de aplicação tenha sido compreendido, deve-se selecionar e preparar os dados para serem analisados na próxima etapa. As seções seguintes tratam desta etapa do processo.

2.3.2 Seleção de Dados

Com um domínio especificado, deve-se localizar e escolher quais as fontes de dados estão relacionadas ao domínio para que um conjunto de dados apropriado possa ser selecionado das mesmas. Por exemplo, se o objetivo é fazer análises das vendas de uma empresa, então as fontes de dados que possuem informações de transações de vendas para clientes, cadastros dos clientes e informações dos analistas de vendas/marketing devem ser consideradas.

As fontes de dados utilizadas para realizar o processo de KDD podem ser classificadas como internas e externas. As fontes internas normalmente são repositórios de dados que já estão incorporados ao sistema de aplicação do domínio em questão, e normalmente, são constituídas por *Data Warehouse*, bases de dados operacionais e dados históricos. As fontes externas são compostas por todos outros tipos de localidades que habitualmente não estão incorporados ao sistema de aplicação como, por exemplo, documentos (e.g. relatórios), Internet (e.g. páginas HTML) e informações do especialista do domínio.

Os dados contidos em fontes internas normalmente já estão armazenados em algum tipo de repositório estruturado (e.g. base de dados relacional) e por isso, requerem menos processamento, pois, eles podem ser recuperados utilizando-se consultas SQL (*Structured Query Language*), por exemplo. Enquanto que os dados contidos em fontes externas precisam ser formatado e inseridos de forma que possam ser recuperados e preparados para análise.

É interessante citar que, freqüentemente, os dados de análise podem nem existir e devem ser armazenados assim que os eventos vão acontecendo, um exemplo disto é apresentado na aplicação *Advanced Scout*, desenvolvida para auxiliar os técnicos dos times da NBA (liga norte-americana de basquete), que armazena os números de um jogo que depois é analisado juntamente com números conseguidos anteriormente (Bhandari et al., 1997).

As fontes podem possuir quantidades de *gigabytes* ou *terabytes* de dados e, além disso, estes dados podem vir de fontes distintas, necessitando que estes sejam reduzidos e colocados num conjunto de dados com estrutura homogênea (e.g. tabela) para permitir a análise conjunta destes. A seção seguinte discute a etapa que trata desses problemas.

Os dados podem estar em diferentes lugares, exigindo que os mesmos sejam integrados. Esta tarefa de integração não é trivial, exigindo a adequação de alguns dados, estas transformações são realizadas nas próximas etapas.

2.3.3 Preparação de Dados

Na etapa de preparação de dados, normalmente, é realizada a atividade de amostragem de dados. Pois nem sempre é possível analisar todos os dados de uma população, sendo necessário pegar uma amostra representativa para realizar o processo de KDD. Os dados também podem estar em formatos diferentes, pois os mesmo, poderão ter sido coletados de diferentes fontes, então, algumas transformações devem ser feitas sobre eles, por exemplo, converter tipos não suportados por uma que todas as fontes consigam representar, se isso for possível.

Outra problemática que deve ser resolvida envolve o tamanho do conjunto de dados, pois a grande quantidade de dados reunida, às vezes, podem impossibilitar a realização do processo de KDD uma vez que alguns algoritmos usados em *Data Mining* (Seção 2.3.5) conseguem tratar apenas um número limitado de registros. Por isso, deve-se utilizar algumas técnicas de amostragem para que se possa reduzir o tamanho do conjunto de dados obtendo um subconjunto que seja relevante e representativo.

A realização de amostragem é crítica dentro do processo de KDD, pois, se não forem utilizadas técnicas estatísticas adequadas, pode-se gerar um subconjunto de dados não representativo, resultando em análises que não demonstram a verdadeira situação dos fatos registrados na base de dados ou distorcendo resultados. Nesse ínterim, devem ser observados os seguintes aspectos para se fazer amostragem dos dados como:

- O tamanho da amostra;
- Estratégias para obtenção da amostra (técnicas estatísticas podem ser utilizadas, bem como o apoio do especialista do domínio);
- Homogeneidade dos dados;
- Dinâmica dos dados (mudanças de valores de atributos ao longo do tempo).

Os dados contidos no subconjunto amostrado podem eventualmente possuir informações replicadas, campos “faltantes” e ruídos (erros). Assim, é preciso limpar e pré-

processar estes dados para que possam ser aplicadas técnicas de *Data Mining*. Esta etapa de adequação dos dados é discutida na seção seguinte.

2.3.4 Limpeza e Pré-processamento de Dados

Após reunir dados de diversas fontes, é muito provável que o conjunto de dados venha a ter registros duplicados, erros nos dados e muitos dados ausentes. E mesmo que se colete dados de apenas uma fonte, estes tipos de problemas ainda podem ocorrer. Por isso, deve-se fazer uma limpeza e pré-processar os dados.

As operações de remoção de ruídos dos dados, tratamento de atributos ausentes e remoção de registros repetidos feitas nesta etapa compreende, entre outros, os seguintes aspectos:

- *Padronização dos valores dos atributos*: em um conjunto de dados constituído por diversas fontes pode acontecer, por exemplo, que o “sexo” possua diferentes valores e tipos com o mesmo significado como “masculino”, “masc”, “m” ou “1”. Então, deve-se transformar estes valores como sendo de um tipo único e iguais para todos;
- *Remoção de registros duplicados*: pode ocorrer que os mesmos dados de um cliente estejam contidos em diferentes registros e sendo considerado como duas pessoas distintas, pois, por exemplo, o nome do cliente em um determinado registro aparece sem abreviações e em outro ele aparece abreviado;
- *Tratamento e eliminação de ruídos*: muitas vezes os dados coletados podem conter erros ocasionados por diversos fatores no momento da recuperação de diversas fontes, como acidentes (e.g. falta de energia elétrica), tipos não suportados pelo gerenciador de base de dados (e.g. objetos multimídia podem não ser suportados e serem armazenados como “lixo”), entre outros. Os campos que contém ruídos nos seus valores devem ser tratados atribuindo o valor correto aos dados ou devem ser eliminados da base de dados caso não tenham como ser tratados;
- *Tratamento de valores ausentes*: é muito comum encontrar registros cujos campos possuem valores ausentes. Isto pode ocorrer, entre outras coisas, devido a erros na entrada dos dados, por exemplo, se no momento em que um operador estiver cadastrando as informações de clientes, ele pode esquecer (ou ignorar) alguns campos de dados. Dessa

forma, deve-se estabelecer critérios para o tratamento de atributos ausentes (Rocha, 1999).

Esta etapa deve ser feita, sempre que possível, com o acompanhamento do especialista do domínio, pois como este possui um conhecimento mais abrangente do domínio em questão, ele poderá ajudar na determinação de quais atributos podem ser descartados ou que devem ser mantidos. Apesar de existirem ferramentas para a execução desta etapa, a participação de um especialista pode torna-la muito mais precisa e confiável.

Segundo Mannila (Mannila, 1996) as etapas apresentadas na Seções 2.3.2, 2.3.3 e 2.3.4 podem corresponder a 80% de todo o tempo gasto durante o processo de KDD. Uma forma de otimizar e simplificar significativamente este trabalho seria a construção de um *Data Warehouse*, uma vez que no projeto de um *Data Warehouse* já deve estar previsto a integração de dados de diferentes fontes, bem como a remoção de ruídos e tratamento de atributos faltantes.

Com os dados selecionados, preparados, limpos e pré-processados pode-se dar início à etapa de *Data Mining* ou mineração de dados. Esta etapa será introduzida na seção seguinte (Seção 2.3.5).

2.3.5 *Data Mining*

A etapa de *Data Mining* ou “mineração” de dados pode ser vista como uma sequência de etapas dentro do processo de KDD, que pode envolver repetidas iterações da aplicação de um determinado algoritmo com o objetivo de se extrair padrões dos dados. É importante frisar que o termo padrão neste trabalho deve ser entendido como sendo uma instanciação de um determinado modelo (Fayyad et al., 1996b; Mitchell, 1997).

Os algoritmos e as técnicas usados para se criar modelos a partir de dados, normalmente, provém de áreas como Aprendizado de Máquina (AM), Reconhecimento de Padrões e Estatística. Estas técnicas, muitas vezes, podem ser combinadas para se obter resultados melhores.

Neste trabalho serão focadas as técnicas de AM, as quais envolvem métodos para aquisição de conhecimento implícito a partir de um conjunto de fatos. Esses métodos são vistos como sistemas de aprendizado que possuem o objetivo de descobrir conhecimento de dados através da construção de um modelo preditor ou descritor. A descrição e a predição

são conseguidas selecionando-se as tarefas, os algoritmos e técnicas de extração de dados.

2.3.5.1 Tarefas de *Data Mining*

Os dois objetivos principais da “mineração” de dados são a previsão e a descrição de modelos. A previsão pode ser conseguida utilizando-se determinadas variáveis para prever valores desconhecidos ou futuros de outras variáveis de interesse. A descrição envolve a descoberta de padrões interpretáveis por humanos que descrevam os fatos cadastrados em bases de dados. Alguns autores (Fayyad et al., 1996b) mencionam que dentro do contexto de KDD, descrever modelos possui maior importância que prever os mesmos embora a maioria dos algoritmos de AM e Reconhecimento de Padrões focarem na previsão.

A previsão e descrição podem ser realizadas através das seguintes tarefas:

- Classificação: a classificação é uma tarefa preditiva que pode ser usada para encontrar um modelo que classifique um item de dado entre várias classes previamente definidas;
- Regras de Associação: responsável por encontrar relações entre os campos de um conjunto de dados;
- Regressão: tarefa preditiva cujo objetivo é encontrar uma função de mapeamento para um item de dado, normalmente numérico, que possibilite a previsão dos valores do mesmo;
- Agregação ou Segmentação: tarefa descritiva cujo objetivo é procurar identificar um conjunto de categorias finitas ou agrupamentos naturais que possam descrever um certo comportamento nos dados. Estas categorias podem ser mutuamente exclusivas ou podem consistir de uma representação hierárquica;
- Resumo ou Síntese: tarefa que visa obter uma descrição compacta de um conjunto de dados para se conseguir fazer análises exploratórias de dados e geração automática de relatórios;
- Modelagem de Dependência: tarefa que visa encontrar um modelo descritivo das dependências significativas entre variáveis. Os modelos de dependência são descritos em dois níveis, no nível estrutural que verifica a dependência entre variáveis e no nível quantitativo que mede a força das mesmas;

- **Evolução:** esta tarefa está focada em descobrir as mudanças mais significativas nos dados através de medidas anteriores ou de valores normativos. O objetivo é modelar o processo gerando uma sequência ou relatando tendências do processo ao longo do tempo.

Com a definição da tarefa de *Data Mining*, pode ser escolhido o algoritmo ou uma combinação deles para serem utilizados. A seção seguinte apresenta alguns dos principais algoritmos de mineração de dados.

2.3.5.2 Algoritmos Usados em *Data Mining*

A escolha de um algoritmo para DM não é uma tarefa trivial, pois, deve-se considerar vários aspectos como o objetivo da tarefa a ser realizada, o tamanho da base de dados, a precisão exigida, entre outros. Muitas vezes, são realizados testes empíricos com diferentes tipos de algoritmos para se descobrir qual deles apresenta o melhor resultado em termos da qualidade e precisão dos padrões encontrados.

Para que o analista possa selecionar qual algoritmo melhor se enquadra a uma determinada tarefa, este deve conhecer quais são os aspectos básicos de cada método em potencial. Os algoritmos de aprendizado de máquina podem ser classificados em vários aspectos: por tipo de aprendizado, por tipo de paradigma, por tipo de linguagem de descrição, e por modo de incorporação de novos exemplos, a Tabela 2.2 apresenta um resumo desta classificação.

Com relação ao modo como esses algoritmos aprendem, eles podem ser classificados como sendo de aprendizado supervisionado e não-supervisionado. No aprendizado supervisionado os algoritmos utilizam dados para treinamento e dados para validação e teste dos modelos gerados. No aprendizado não-supervisionado, não existem dados de treinamento, assim, o algoritmo deve tentar encontrar os padrões através de caracterização e/ou segmentação destes dados.

Os algoritmos de aprendizado de máquina podem ser de diferentes paradigmas, tais como:

- *Paradigma Simbólico:* o aprendizado é feito na tentativa de se construir representações simbólicas de um conceito através da análise de exemplos e contra-exemplos desse conceito. As representações simbólicas estão tipicamente na forma de alguma expressão lógica, árvore de decisão, regras de produção, rede semântica, etc. Os métodos de

aprendizado simbólico podem ser tratados como: proposicional e relacional (Tecuci and Kodratoff, 1995).

- *Paradigma Estatístico*: as técnicas estatísticas, em geral, tendem a focar tarefas em que todos os atributos têm valores contínuos. Vários modelos matemáticos são utilizados para se construir os modelos dos dados (Elder IV and Pregibon, 1996).
- *Paradigma Instance-based*: uma forma de classificar um caso é lembrar de um caso similar cuja classe é conhecida e assumir que o novo caso terá a mesma classe. Esta filosofia exemplifica os sistemas *instance-based*, que classificam casos nunca vistos através de casos similares conhecidos (Quinlan, 1993; Aha et al., 1991).
- *Paradigma Conexionista*: as redes neurais artificiais (RNA) são construções matemáticas relativamente simples que utilizam o mecanismo de paralelismo, onde são conectados um grande número de pequenas unidades de processamento ligadas em rede. As RNA possuem a capacidade de aprender por exemplos e fazer interpolações e extrapolações do que aprenderam. No paradigma conexionista não se procura obter regras como na abordagem simbólica, mas determinar a intensidade de conexões entre neurônios (Braga et al., 2000).
- *Paradigma Genético*: um classificador genético consiste de uma população de elementos de classificação que competem para fazer uma predição, onde os elementos que possuem uma performance fraca são descartados e os mais fortes proliferam, produzindo variações de si mesmos. Os algoritmos deste paradigma são conhecidos por algoritmos genéticos, os quais baseiam-se nos mecanismos de seleção natural e genético para fazer otimizações e buscas (Goldberg, 1989).

Em um sistema de aprendizado é preciso representar exemplos, conceitos ou a teoria do domínio da aplicação na forma de linguagens de descrição. Estas linguagens são:

- *Linguagem de descrição de exemplos ou instâncias*: descreve os exemplos utilizados pelo programa para aprender conceitos, estabelecendo limites sobre os tipos de padrões que o sistema pode aprender.
- *Linguagem de descrição de hipóteses ou conceitos aprendidos*: descreve o estado interno de um programa de aprendizado, correspondente a teoria dos conceitos ou padrões que

existem nos dados, estabelecendo limites sobre o que pode ou não pode ser aprendido (e.g. regras de decisão, árvores de decisão).

- *Linguagem de descrição da teoria de domínio ou conhecimento de fundo*: descreve todo o conhecimento prévio que o programa possui a respeito do domínio.

Os algoritmos de aprendizado indutivo também podem ser classificados de acordo com a possibilidade de integração de novos exemplos aos modelos encontrados, podendo ser classificados como:

- *Não Incremental*: necessita de que todos os exemplos de treinamento, simultaneamente, estejam disponíveis para que seja induzido um conceito. É vantajoso usar esses algoritmos para problemas de aprendizado onde todos os exemplos estão disponíveis e, provavelmente, não irão ocorrer mudanças.
- *Incremental*: revê a definição do conceito corrente, se necessário, em resposta a cada nova instância de treinamento observada. Os exemplos observados são considerados um a um pelo sistema, isto é, o sistema considera o primeiro exemplo e, de acordo com esse exemplo, constrói uma determinada hipótese; a seguir considera um segundo exemplo, que pode ou não modificar a primeira hipótese, baseando-se em como esta classifica o segundo exemplo.

Na Tabela 2.2 é apresentado um resumo das características gerais descritas dos sistemas de aprendizado de máquina.

Tipo de Aprendizado	Paradigmas de Aprendizado	Linguagens de Descrição	Modo que os Novos Exemplos são Integrados
Supervisionado	Simbólico	Exemplo ou Instâncias	Incremental
Não Supervisionado	Estatístico	Hipóteses ou Conceitos Aprendidos	Não Incremental
	Instance-Based	Teoria de Domínio ou Conhecimento de Fundo	
	Conexionista Genético		

Tabela 2.2: Classificação dos algoritmos de AM usados em *Data Mining*.

Alguns dos algoritmos utilizados para fazer extração de padrões (Kohavi et al., 1994; Tecuci and Kodratoff, 1995) são listados na Tabela 2.3. Estes algoritmos foram classificados

em função de tipo de aprendizado, paradigma de aprendizado, linguagem de descrição e modo que novos exemplos são incorporados.

Indutor	Tipo	Paradigma	Linguagem	Incrementabilidade
aha-IB	Supervisionado	Instance-based	Exemplos	Incremental
C4.5	Supervisionado	Proposicional	Árvore de decisão	Não incremental
CART	Supervisionado	Estatístico	Árvore de decisão	Não incremental
CN2	Supervisionado	Proposicional	Regras de produção	Não incremental
Decision Tables	Supervisionado	Instance-based	Exemplos	Incremental
ID3	Supervisionado	Proposicional	Árvore de decisão	Não incremental
Naive Bayes	Supervisionado	Estatístico		
Nearest-neighbor	Não supervisionado	Estatístico		Incremental
OC1	Supervisionado	Proposicional	Árvore de decisão	Não incremental
Perceptron	Supervisionado	Conexionista		Não incremental
T2	Supervisionado	Proposicional	Árvore de decisão	Não incremental
Foil	Supervisionado	Relacional	Cláusulas de Horn	Não incremental
Golem	Supervisionado	Relacional	Cláusulas de Horn	Não incremental

Tabela 2.3: Exemplos de algoritmos de Aprendizado de Máquina.

2.3.5.3 Extração de Padrões

Uma vez que a tarefa de *Data Mining* foi definida e o algoritmo foi escolhido, pode-se iniciar o processo de extração de conhecimento. Entretanto, deve ser observado que o analista precisa observar alguns aspectos de extração de conhecimento, tais como:

- *Definição de Parâmetros*: algoritmos diferentes, normalmente, possuem parâmetros distintos. A definição correta dos parâmetros, freqüentemente, pode ser conseguida através de experimentos empíricos. Ou seja, quanto mais experiente for o analista maior serão as chances deste conseguir ajustar os parâmetros adequadamente.
- *Redução de Atributos*: pode ser percebido que alguns dos atributos do conjunto de dados não possuem muita relevância para o algoritmo selecionado (e.g. atributos numéricos para algoritmos simbólicos), então, pode-se pensar em se excluir tais atributos.
- *Generalização*: um algoritmo deve ser capaz de reconhecer padrões desconhecidos e não decorar exemplos apresentados. Para isso, deve-se utilizar técnicas estatísticas que reforçam a generalização de um preditor ou descritor, além de garantir pequenas taxas de erros. Para isso, as amostras são divididas em conjuntos separados de treinamento e teste. Possibilitando, dessa forma, medir a taxa de erro através do conjunto de teste. Essas técnicas são:

- *Holdout*: o método *holdout* divide fixamente a amostra de exemplos em 2/3 (dois terços) para treinamento e 1/3 (um terço) para teste. A estimativa deste método é adequada para grandes conjuntos de dados.
- *Resampling*: os métodos de *resampling* realizam experimentos com diferentes partições das instâncias em teste e treinamento e possui como taxa de erro a média das taxas de erros calculadas sobre o conjunto de teste em cada experimento. Um método bastante utilizado de *resampling* é o *cross-validation* que divide um conjunto de exemplos em k partições mutuamente exclusivas. A cada iteração, é utilizada 1 partição para testar o sistema de aprendizado e $k-1$ restantes para o treinamento. Como são realizadas k iterações, então todas as k partições servem como conjunto de teste.
- *Mudanças dos dados*: dados que mudam com muita frequência podem invalidar os padrões descobertos anteriormente. A possível solução para este problema é incluir métodos incrementais que consigam atualizar os padrões já encontrados;
- *Compreensão de padrões*: em muitas aplicações é importante tornar as descobertas mais facilmente compreensíveis aos usuários. Para tanto, podem ser utilizadas técnicas sofisticadas de visualização de dados. Os resultados obtidos, normalmente, são apresentados em algum tipo de representação como árvores de decisão, regras de decisão, gráficos ou relatórios escritos.

Várias áreas estão envolvidas no processo de KDD desde a coleta dos dados a serem analisados até a visualização dos resultados obtidos. A próxima seção apresenta as principais áreas relacionadas ao processo de Extração de Conhecimento de Base de Dados.

2.3.6 Interpretação e Avaliação de Padrões

Os padrões encontrados na etapa de *Data Mining* devem ser validados a partir da interpretação e avaliação destes. Os usuários envolvidos devem interpretar os padrões extraídos, e para isto, podem lançar mão de ferramentas estatísticas e de visualização que permitam fazer uma “leitura” precisa sobre os resultados obtidos, de forma a possibilitar a verificação da validade e novidade, ou mesmo, a irrelevância dos padrões encontrados.

É importante notar que esta tarefa deve ser feita em conjunto pelos usuários envolvidos no processo de extração de conhecimento, pois somente com o auxílio de especialistas é que um analista terá condições de avaliar se o que foi descoberto é um conhecimento relevante ou não.

Caso o conhecimento não seja validado, então, provavelmente, deve-se retornar às etapas anteriores e tentar refazer-las ou melhorá-las. Esta iteração pode ocorrer até que se obtenha resultados aceitáveis ou concluir-se que não seja possível extrair conhecimento relevante dos dados.

2.3.7 Consolidação do Conhecimento

O conhecimento depois de validado deve ser consolidado, ou seja, ele deve agregar valor a um sistema de apoio à tomada de decisão. Isto é feito incorporando-se tal conhecimento a um sistema de aplicação ou, simplesmente, documentando-o e relatando-o às partes interessadas. Após a consolidação do conhecimento, pode-se resolver conflitos potenciais entre os conhecimentos anteriores e previamente extraídos ou acreditados.

Para que o conhecimento extraído seja realmente utilizado todos os usuários que fazem parte do processo decisório devem receber todas as informações sobre novas descobertas encontradas. Para isso, deve-se possuir ferramentas que garantam um grande fluxo de troca de informações e boa comunicação entre eles.

2.4 Áreas de Apoio

O processo de KDD é feito incorporando-se várias técnicas de diferentes áreas como Aprendizado de Máquina, *Data Warehousing*, Estatística e Visualização de Dados. Entretanto, neste trabalho, a etapa de extração de padrões (*Data Mining*) é vista como sendo uma tarefa inerente à área de Aprendizado de Máquina, e as demais são consideradas como áreas de apoio ao processo de KDD. As áreas que oferecem suporte ao KDD são apresentadas nas próximas seções.

2.4.1 *Data Warehousing*

Data Warehousing é um processo, não um produto, para montar e gerenciar dados de várias fontes com o propósito de se ganhar uma visão detalhada e singular de parte ou do todo de um negócio. O grande produto gerado de um projeto de *Data Warehousing* é o seu *Data Warehouse*.

Segundo Inmon (Inmon, 1996) para se realizar o processo de KDD não é necessário que se tenha implementado um *Data Warehouse*, porém, se uma organização qualquer resolver realizar um processo de extração de conhecimento num determinado domínio de aplicação, deve ser destacado que se esta organização possuir um DW com seus dados, grande parte do tempo que deveria ser gasto durante as etapas de processamento (seleção, preparação e limpeza) dos dados seria reduzido drasticamente, permitindo, dessa forma, que os analistas focassem a atenção em outras tarefas cruciais como *Data Mining*, Avaliação e Consolidação de Conhecimento.

O KDD não substitui o *Data Warehousing*, mas ao contrário, ele vem com o intuito de aprimorar o processo de apoio à tomada de decisão a partir do conhecimento extraído automaticamente dos dados.

2.4.2 Estatística

A Estatística é a área da matemática que estuda a coleta, organização e interpretação de dados numéricos, especialmente a análise de características da população por inferências a partir de amostras. As técnicas de estatística possuem muita importância dentro do processo de KDD e boa parte dos métodos utilizados em KDD tiveram suas origens dentro da Estatística (Elder IV and Pregibon, 1996; Glymour et al., 1997).

As áreas de KDD e da Estatística estão fortemente ligadas. Ambas as disciplinas tem como objetivo encontrar padrões e regularidade nos dados. Em geral, a etapa de *Data Mining*, enfatiza mais a facilidade de entendimento do conhecimento adquirido que basicamente a precisão. Além disso, a maioria dos algoritmos usados em DM está mais voltada para a produção de conjuntos de declarações sobre dependências locais entre variáveis de interesse do que propriamente na construção de modelos globais que incluam todas as variáveis de interesse do problema (Glymour et al., 1997).

As técnicas estatísticas auxiliam diversas etapas do processo de KDD, como na se-

leção, preparação e limpeza dos dados, além de grande parte dos algoritmos de aprendizado de máquina fazerem uso de mecanismos disponíveis na estatística para realizar a descoberta de padrões, calcular aproximações, médias, taxas de erros e desvios (Elder IV and Pregibon, 1996; Rocha, 1999).

2.4.3 Visualização de Dados

As técnicas e ferramentas para visualização de dados são “instrumentos” indispensáveis ao processo de KDD. Elas podem ser usadas durante a execução das etapas do processo de extração de conhecimento melhorando a compreensão dos resultados obtidos e até a comunicação entre os usuários (Rezende et al., 1998b).

As técnicas de Visualização de Dados estimulam naturalmente a percepção e inteligência humana, aumentando a capacidade de entendimento e associação de novos padrões. Logo, a visualização de dados utiliza a percepção humana como um primeiro método para descobrir valores. Poderosas ferramentas de visualização que consigam gerar visualizações em diversas metáforas (árvore, regras, gráficos 3D/2D, espectro) combinadas com técnicas de *Data Mining* podem melhorar em muito o processo de KDD (Oliveira and Rezende, 1998).

2.5 Alguns Desafios da Área de Extração de Conhecimento de Base de Dados

Como KDD é uma área relativamente recente, diversas dificuldades e desafios estão surgindo continuamente, os quais precisam ser atacados, dessa forma, esta seção pretende exemplificar alguns dos principais problemas e direções futuras para o processo de KDD.

Alguns importantes desafios que os pesquisadores de KDD podem enfrentar, ou já estão enfrentando, são (Uthurusamy, 1996):

- *Interação com usuário*: muitas das tecnologias de KDD existentes atualmente não são realmente interativas e não conseguem incorporar facilmente o conhecimento prévio a respeito de um domínio de aplicação. O uso do conhecimento anterior (Pazzani and Kibler, 1992) relevante é de grande importância para o processo de KDD. Pode-se tentar incorporar um sistema especialista para tentar colher o conhecimento de especialistas em uma base de conhecimento (Rezende and Pugliesi, 1998).

- *Integração com outros sistemas*: um sistema isolado de descobrimento pode não possuir muita utilidade caso este não possa ser integrado a outros sistemas como gerenciadores de bases de dados, ferramentas analíticas e de visualização.
- *Suporte a novas tecnologias de base de dados*: com a evolução da tecnologia de armazenamento, os dados armazenados passarão a conter além de textos e números, objetos gráficos, multimídia, bem como dados não-estacionários (alteração contínua), temporais, entre outros. Gerenciadores de base de dados orientados à objetos podem tratar este tipo de problema de armazenamento, facilitando a geração de metadados.
- *KDD em um ambiente de rede e distribuído*: o rápido crescimento de recursos disponíveis na Internet demanda uma grande necessidade por pesquisas para o desenvolvimento de ferramentas, técnicas e sistemas que possam permitir a realização do processo de KDD neste ambiente conectado e distribuído. Ainda, a tendência da área de KDD é guiar-se para um descobrimento de conhecimento “colaborativo” envolvendo um grande time de analistas e especialistas do domínio espalhados que utilizarão bases de dados distribuídas pela rede. As pesquisas atuais de agentes inteligentes é um começo para se conseguir atingir os desafios impostos à área de KDD pelas novas tecnologias de WWW (*World Wide Web*) e base de dados multimídia.

2.6 Considerações Finais

Sistemas capazes de auxiliar a aquisição de conhecimento para serem na solução de problemas são desejados produtos da ciência da Computação. A descoberta de novos e inesperados fenômenos podem trazer benefícios em diversos campos da ciência fornecendo um maior entendimento dos seus respectivos domínios (Wiederhold, 1996).

Várias técnicas são usadas na tentativa de se extrair conhecimento de fatos. Pode-se extrair conhecimento de especialistas ou através dos dados. A extração de conhecimento através de especialistas (explícita) visa sintetizar o conhecimento de um especialista humano modelando-o para que este seja incorporado a um sistema. Já a extração de conhecimento de dados (implícita), é uma forma automática de se extrair conhecimento a partir de dados mantidos em alguma estrutura de armazenamento utilizando-se técnicas e ferramentas para encontrar modelos e padrões.

A Extração de Conhecimento de Base de Dados (KDD) é uma área da ciência de computação que visa extrair padrões e modelos de grandes quantidades de dados. Os padrões podem ser extraídos realizando-se alguns passos como seleção, preparação, limpeza e pré-processamento dos dados, “mineração” de dados, avaliação e consolidação do conhecimento.

Como qualquer área em pleno desenvolvimento, novos desafios devem ser continuamente superados. No contexto deste trabalho, é focado o estudo das dificuldades encontradas na integração de ferramentas que auxiliam a descoberta de conhecimento e a execução do processo de KDD num ambiente de redes de computadores. Enfim, este trabalho propõe um sistema que pode tratar alguns destes problemas fazendo uso de técnicas que possibilitem a realização do processo de KDD num ambiente heterogêneo e distribuído com o apoio de Agentes de Software. Dessa forma, uma revisão sobre agentes é apresentado no próximo capítulo.

Capítulo 3

Agentes de Software

A natureza dinâmica e distribuída de algumas aplicações exige que os aplicativos de software não apenas respondam às requisições de informação, mas também que estes consigam antecipar, adaptar e buscar novas formas para auxiliar o usuário. Dessa forma, os projetos de sistemas computacionais tendem a tornar-se cada vez mais complexos uma vez que estes precisam, por exemplo, especificar interfaces entre diferentes aplicativos, possuir características distribuídas e de persistência. Estes sistemas também devem dar suporte na execução e gerenciamento de programas distribuídos (Bradshaw, 1997a).

A área de Agentes de Software (AS) é integrada por pesquisadores de diferentes áreas, como, Inteligência Artificial, Sistemas Distribuídos, Interface Humano Computador e Robótica, que juntos têm como principais objetivos: atender aos novos requisitos exigidos por determinadas aplicações, facilitar a interação usuário/máquina e a construção de sistemas inteligentes distribuídos.

O nome Agente de Software, é considerado como sendo resultante das pesquisas desenvolvidas pelas comunidades de agentes inteligentes e de agentes de software. A primeira está envolvida com a construção de sistemas inteligentes capazes de reagir a eventos tanto em ambientes do mundo real (e.g. robôs) quanto do mundo virtual (e.g. programas). Enquanto que a outra está preocupada com o desenvolvimento de programas de software mais confiáveis, menores e com a propriedade de se mover dentro de uma rede de computadores distribuindo informações para outros agentes que podem ser tanto humanos quanto aplicações de software (Hendler, 1996).

Este capítulo apresenta uma visão geral sobre os Agentes de Software, mostrando

algumas definições, arquiteturas e linguagens utilizadas, bem como algumas das possíveis aplicações dos agentes.

3.1 Definição

A partir das definições observadas nos trabalhos de alguns pesquisadores é possível perceber que vários autores possuem diferentes idéias sobre a área de AS. Dessa forma, não existe uma definição aceita por unanimidade pelos pesquisadores sobre o que seja exatamente um agente de software, porém, muitas destas definições estudadas complementam-se. Nesta seção é apresentado um resumo das definições encontradas.

Russel e Norvig (Russel and Norvig, 1995) definem o termo agente como: “um agente é tudo que pode ser visto como percebendo seu ambiente através de sensores e agindo sobre ele através de efetadores”. Realizando uma analogia simples, um humano possui diversos sensores como olhos, ouvido e nariz, e efetadores como braços, pernas e boca. Um robô possui, por exemplo, câmeras de vídeo, antenas de rádio e fontes de raios infravermelhos como sensores e motores como sendo os efetadores. Enquanto que um programa tem suas percepções e ações codificadas em uma seqüência de bits.

Maes (Maes, 1994b) e Hendler (Hendler, 1996) mencionam sobre agentes inteligentes como programas de IA cuja finalidade é agir em diversos ambientes de importância para os humanos, sendo divididos em duas categorias: agentes físicos e agentes de informação. Os agentes físicos trabalham em um ambiente onde seja difícil colocar um ser humano (e.g. espaço) ou que seja perigoso (e.g. núcleo de um reator nuclear). Os agentes de informação, ou *softbots* (*software robots*), atuam em um mundo virtual onde exista uma grande quantidade de informações espalhadas por diversos computadores (e.g. *Internet*).

Wooldridge e Jennings (Wooldridge and Jennings, 1994) distinguem dois usos do termo agentes:

- *Noção “fraca” de agência*: o termo agente é usado, geralmente, para denotar tanto hardware quanto software que possuem as seguintes propriedades:
 - *Autonomia*: agentes operam sem a intervenção direta de humanos ou outros agentes, e possuem algum tipo de controle sobre suas ações e estados internos;

- *Sociabilidade*: agentes interagem com outros agentes (ou humanos) através de algum tipo de linguagem de comunicação compreensível por ambas as partes;
 - *Reatividade*: agentes percebem o ambiente (físico ou virtual) e reagem em tempo às mudanças ocorridas nele;
 - *Pró-atividade*: agentes não respondem apenas às mudanças no seu ambiente, eles ainda podem exibir um comportamento orientado à objetivo, ou seja, ele também pode tomar a iniciativa.
- *Noção “forte” de agência*: uma visão mais especializada atribuída, principalmente, pelos pesquisadores envolvidos em IA, definem agentes como sendo sistemas de computador que além de possuírem as propriedades anteriormente citadas são conceituados e implementados utilizando-se conceitos que normalmente são aplicados aos humanos. Por exemplo, é muito comum em Inteligência Artificial caracterizar um agente usando noções mentais, como conhecimento, crença, intenção e obrigação.

Bradshaw (Bradshaw, 1997a) acrescenta outras propriedades que os agentes devem possuir para se diferenciar de um simples programa de computador:

- *Capacidade de Inferência*: agentes podem agir sobre especificações abstratas de tarefas utilizando um conhecimento anterior, conseguindo ir além das informações fornecidas, e devem possuir algum modelo de si próprio, de usuários, de situações e de outros agentes;
- *Continuidade*: agentes conseguem fazer persistir a sua identidade e estados durante longos períodos de tempo;
- *Adaptabilidade*: agentes são capazes de aprender e melhorar com a experiência;
- *Mobilidade*: agentes podem migrar de forma intencional de um determinado local para outro.

Uma outra abordagem para se definir os agentes de software é diferenciar um agente de um simples programa de computador. Dessa forma, apesar de existirem diferentes definições sobre agentes, ao menos torna-se possível estabelecer critérios os quais serão tomados como válidos para o restante deste trabalho.

Adotando as definições de (Franklin and Graesser, 1997), neste trabalho estão sendo consideradas as seguintes propriedades para caracterizar um agente de software:

- reatividade: capacidade de responder (sentir e reagir) em tempo às mudanças sofridas no ambiente;
- autonomia: executa suas ações sem a necessidade de intervenção e exerce controle sobre suas ações;
- pró-atividade: não reage simplesmente em resposta às mudanças no ambiente, mas consegue tomar a iniciativa;
- persistência: é um processo que consegue executar continuamente (guarda seu estado);
- mobilidade: capacidade de se auto-transportar de uma máquina a outra.

Deve ser ressaltado que todos os agentes de software são programas, porém, nem todos os programas podem ser considerados como agentes. Como exemplo, pode-se assumir que um corretor ortográfico embutido em um processador de textos não é um agente pois não possui as propriedades definidas acima. Uma vez que é o usuário quem aciona o corretor, seleciona o tipo de correção e o corretor não armazena as correções feitas anteriormente quando o processador de texto termina, este é considerado como um programa. Porém, se o corretor ortográfico consegue perceber o usuário digitando (reatividade) e realizar a checagem ao mesmo tempo sugerindo correções (autonomia), ou lançando mensagens de dicas em função do tipo de texto (e.g. carta) que o usuário escreve (pró-atividade) e salvando todas as tarefas de correções feitas para o usuário visando corrigir automaticamente erros iguais (persistência), este programa pode ser considerado como um agente de software segundo (Franklin and Graesser, 1997). Ainda, o corretor ortográfico pode ser capaz de buscar dicionários em diferentes servidores de dicionários (mobilidade), por exemplo.

Os agentes de software podem possuir algumas das “habilidades” citadas, como, mobilidade, reatividade, autonomia. Logo, eles podem ser classificados em função de suas características peculiares. Ainda, os agentes podem estar contidos em vários ambientes distintos, onde cada qual possui um comportamento particular. Dessa forma, os tipos de agentes e os possíveis ambientes são discutidos na seção seguinte.

3.2 Tipologia de Agentes e de Ambientes

A investigação dos possíveis tipos de agentes de software é apresentada nesta seção. O objetivo de separar os agentes pertencentes a classes distintas é necessário para compreender melhor o que são essas entidades. Dessa forma, algumas classes de agentes encontrada na literatura são apresentadas, ou seja, o objetivo é investigar a tipologia (subseção 3.2.1) dos agentes. Uma tipologia refere-se ao estudo dos tipos de entidades. Os agentes de software podem existir em diversos tipos de ambientes, então, alguns dos possíveis ambientes de agentes também são citados na subseção 3.2.2.

3.2.1 Tipos de Agentes de Software

Para que seja possível classificar tipos de agentes de software faz-se necessário que sejam estabelecidos alguns critérios e características particulares que permitam a realização de tal classificação. Neste ínterim, Nwana (Nwana, 1996) apresentou um estudo selecionando algumas propriedades que facilitam tal tarefa. Na concepção apresentada os agentes podem ser classificados de acordo com suas habilidades, tais como:

- *Mobilidade*: se um agente possuir a capacidade de transportar-se através de uma rede de computadores armazenando seu estado anterior, então ele é considerado um agente móvel e estático, caso contrário.
- *Comportamento*: a existência de um modelo simbólico interno de raciocínio que permita ao agente planejar e negociar com outros agentes na execução de tarefas de forma coordenada, classifica o agente como sendo deliberativo. Enquanto que os agentes ditos reativos não possuem nenhum modelo simbólico interno de seu ambiente e, ao invés disto, eles reagem a estímulos vindos do seu ambiente.
- *Exibição de Atributos Primários*: as propriedades de autonomia, cooperação e aprendizagem classificam os agentes como sendo colaborativos, de interface ou inteligentes.
- *Papel*. Os agentes podem ser classificados através de seus papéis, como, por exemplo, os agentes de Internet que têm como papel principal a busca por informações na *Internet*.
- *Filosofia Híbrida*: os agentes podem possuir duas ou mais filosofias de agentes ao mesmo tempo.

As características apresentadas cobrem grande parte das habilidades que os agentes conseguem apresentar. Essas propriedades possibilitam a definição de uma tipologia de agentes de software. Dessa forma, algumas das categorias mais comuns de agentes existentes são:

- *Agentes Colaborativos*: agentes que enfatizam a autonomia e cooperação com outros agentes para realizar tarefas. Eles podem aprender, porém, este aspecto não é muito relevante para este tipo de agente. Para que haja uma coordenação entre os agentes, estes devem ser capazes de negociar (Nwana, 1996).
- *Agentes de Interface*: agentes que enfatizam a autonomia e o aprendizado para realizar as suas tarefas, metaforicamente, eles agem como se fossem assistentes pessoais que estejam colaborando com o usuário no mesmo ambiente de trabalho. A colaboração existente difere da apresentada pelos agentes colaborativos, uma vez que não é necessário que exista uma linguagem formal explícita para a comunicação como é o caso da comunicação intra-agentes (Maes, 1994a).
- *Agentes Móveis*: são programas autônomos capazes de migrar de uma máquina para outra em uma rede heterogênea de computadores (e.g. *Internet*). Em outras palavras, o agente que esteja executando em um computador *A* consegue suspender sua execução em qualquer ponto, transportar seu código e estado para um computador *B* e terminar sua execução podendo voltar em seguida para a máquina de origem ou seguir para uma outra. Esta migração para uma fonte eletrônica de recursos permite ao agente acessar localmente os recursos disponíveis, eliminando, assim, a transferência dos dados pela rede (Gray, 1997).
- *Agentes de Informação/Internet*: são agentes capazes de “navegar” por redes heterogêneas de computadores (e.g. *Internet*) para localizar, combinar, processar, organizar e trazer informações requisitadas por usuários ou programas (Knoblock and Ambite, 1997).
- *Agentes Reativos*: são agentes que respondem a estímulos enviados pelo seu ambiente operando de forma autônoma em tarefas específicas (e.g. sensorimento). Entretanto, nenhuma especificação de comportamento é previamente definida, ou seja, eles não

possuem um sistema de planejamento, sistema de aprendizagem e máquinas de inferência como em alguns sistemas clássicos de IA. Normalmente, a comunicação entre estes agentes não é feita via *broadcasting* de mensagens, e sim por transmissões realizadas “agente a agente”, que tende a ser minimizadas permitindo que cada agente tome decisões por si próprios tornando suas respostas à estímulos mais eficiente (Maes, 1994b; Nwana, 1996).

- *Agentes Inteligentes*: os agentes inteligentes baseiam-se nos sistemas de planejamento e de raciocínio lógico de Inteligência Artificial. Estes agentes possuem representações do seu ambiente e inferem a respeito dos eventos que ocorrem dentro dele. O componente principal deste agente é sua base de conhecimento, que é utilizada para formar representações do mundo em que vive e que são utilizadas por processos de inferência para derivar novas representações permitindo que sejam realizadas deduções sobre o que deve ser feito. Estes agentes conseguem adaptar-se às mudanças ocorridas em seu meio, e assim, conseguem aceitar novas tarefas (Russel and Norvig, 1995).
- *Agentes Híbridos*: são agentes que possuem características existentes em diversos tipos de agentes.

Os agentes também podem ser classificados a partir das suas funcionalidades em aplicações do mundo real, simplificando, dessa forma, a tipificação dos mesmos. Pode-se separar dois objetivos principais na utilização de agentes de software (Bradshaw, 1997a):

- *Simplificação da complexidade da computação distribuída*: a tentativa de aumentar o nível de abstração existente dos detalhes de hardware, software e padrões de comunicação através da troca da interface atual “programa-programa” por uma interface mais poderosa, geral e uniforme baseada em “agente-agente”.
- *Superação das limitações atuais das abordagens de interface ao usuário*: a tentativa de aumentar o nível de abstração existente das interfaces “humanos-programas” (entre usuário e software) através da delegação de tarefas complexas para os agentes deixando todos os detalhes das especificações para executar tais tarefas a cargo do agente.

Os agentes devem conviver em algum tipo de ambiente, os quais também podem possuir características diferentes. Estas propriedades dos ambientes dos agentes estão apresentadas na próxima seção.

3.2.2 Tipos de Ambientes

A relação entre os agentes e o ambiente onde residem, normalmente, é vista da seguinte forma: as ações são executadas pelos agentes em respostas aos eventos que estes conseguem perceber do ambiente. Os tipos de ambiente onde os agentes podem existir podem ser descritos em termos de (Russel and Norvig, 1995):

- *Acessibilidade*: se o aparato sensorial de um agente fornece a ele acesso ao estado completo do ambiente, então o ambiente é dito acessível. Um ambiente é eficientemente acessível se os sensores do agente detectam todos os aspectos que são relevantes para a escolha de uma ação. Um ambiente inacessível às vezes é conveniente porque o agente não precisa manter todos os possíveis estados internos do seu mundo.
- *Previsibilidade*: se o próximo estado do ambiente é totalmente determinado pelo estado atual e ações selecionadas pelos agentes, então o ambiente é considerado determinístico. Caso contrário ele é não-determinístico.
- *Episódios*: num ambiente episódico, a experiência de um agente é dividida em episódios. Sendo que cada um consiste do agente percebendo e então agindo. A qualidade das ações tomadas pelos agente depende apenas episódio atual, uma vez que os episódios subsequentes não dependem das ações tomadas em episódios anteriores.
- *Variabilidade*: se um ambiente pode modificar-se enquanto um agente está deliberando, então o ambiente é dito como dinâmico para o agente, de outra forma ele é considerado estático. Ambientes estáticos são menos complexos para os agentes pois eles não precisam ficar olhando para o seu mundo enquanto estão decidindo qual ação tomar.
- *Tamanho*: se existe um número limitado de ações e percepções claramente definidas, então o ambiente é chamado de discreto. Em contrapartida, num ambiente contínuo, este número é alterado a todo o momento, e assim, o agente deve conseguir tratar essas mudanças tomando decisões corretas para as mesmas.

A classificação de agentes e ambientes é necessária para se entender estas entidades computacionais. Um outro aspecto importante que deve ser levado em consideração diz respeito ao projeto de agentes de software, que é tratado na próxima seção.

3.3 Projeto de Agente de Software

Para se construir um sistema de agentes de software ou mesmo um único agente é necessário verificar qual a é melhor arquitetura e linguagem de construção existente. As próximas subseções apresentam uma visão geral sobre tais aspectos de projeto.

3.3.1 Arquiteturas de Agentes

De uma forma bastante genérica pode-se dizer que as arquiteturas definidas nos projetos de agentes estão contidas dentro de três tipos (Wooldridge and Jennings, 1994; Hendler, 1996):

- *Arquiteturas Deliberativas*: considera um agente de software como sendo um sistema baseado em conhecimento. Esta arquitetura é pertencente ao ramo da Inteligência Artificial simbólica, que possui a hipótese de que estas entidades possuem a capacidade de realizar tarefas inteligentes. E para isso, alguns tópicos como sistemas de planejamento, processos deliberativos e comportamentos de agentes que são guiados por atitudes mentais (e.g. crenças) devem ser considerados. Este tipo de arquitetura é bastante fascinante, porém, a sua realização pode ser muito complexa ou até inviável.
- *Arquiteturas Reativas*: diferentemente da primeira, esta arquitetura, criada por pesquisadores frustrados com os resultados do paradigma de símbolos, não possui nenhum tipo de modelo simbólico central e não utiliza raciocínio simbólico complexo, mas, ao invés disto, este tipo de abordagem preocupa-se com a construção de agentes que reagem às mudanças ocorridas em seu ambiente.
- *Arquiteturas Híbridas*: a utilização de arquiteturas totalmente deliberativas ou reativas pode não apresentar a melhor solução para se construir agentes, assim, pode ser interessante utilizar arquiteturas que unam os conceitos de diferentes paradigmas, conhecidas por híbridas.

3.3.2 Linguagens de Agentes

Uma linguagem de agentes é vista como sendo uma ferramenta que permite a programação de sistemas computacionais de hardware e software em termos dos conceitos existentes sobre os agentes. Para isso, ela deve permitir que atributos de agentes como crença, objetivos, ou

noções mentais, possam ser representados nos programas de agentes. Exemplos de linguagens para construção de agentes são: *Telescript General Magic*, *Agent-Oriented Programming* (Shoham, 1997), Aglets da IBM, entre outras.

Para que um projeto de AS seja bem sucedido em aplicações do mundo real, os agentes devem conseguir trocar informações entre outros agentes e sistemas envolvidos no seu ambiente. Assim, a próxima seção trata a respeito da comunicação entre os agentes de software.

3.4 Comunicação Entre Agentes

A definição do que seja um agente de software e quais tipos são considerados como válidos ainda gera conflitos entre os pesquisadores da área. Todavia, Finin (Finin et al., 1997) afirma que a capacidade de interação e a interoperabilidade é indiscutivelmente desejável pela maioria destes. O fundamento essencial para uma interação inteligente é o compartilhamento de conhecimento, que envolve o entendimento mútuo do conhecimento e a comunicação deste conhecimento.

Para que os agentes de software possam interagir e interoperar eficientemente, são necessários três componentes fundamentais: linguagem comum entendida por todos os agentes; compreensão comum do conhecimento sendo transmitido e habilidade de transmitir qualquer informação que esteja incluída nos dois primeiros componentes citados. Dessa forma, a interação entre agentes pode ser interpretada como sendo a comunicação de informações e conhecimentos que sejam mutuamente entendidos.

O consórcio conhecido por KSE (*Knowledge Sharing Effort*) foi criado para incentivar o desenvolvimento de uma infra-estrutura técnica para suportar o compartilhamento de conhecimento entre sistemas. Esta organização é formada por quatro grupos de trabalho, onde são feitas pesquisas sobre os problemas encontrados nas tecnologias atuais de representação de conhecimento (Neches, 1992).

O primeiro grupo conhecido por *Interlingua* desenvolve uma linguagem comum para expressar o conteúdo de uma base de conhecimento, denominada KIF (*Knowledge Interchange Format*) que pode ser utilizada para apoiar a tradução do conteúdo de uma linguagem para outra, ou como uma linguagem comum entre dois agentes que possuem diferentes linguagens de representação.

O segundo grupo, chamado KRSS (*Knowledge Representation System Specification*), preocupa-se com a definição de estruturas comuns dentro de famílias de linguagem de representação. Um terceiro grupo, chamado SRKB (*Shared Reusable Knowledge Bases*), preocupa-se em facilitar o consenso sobre base de dados compartilháveis (ontologias).

Finalmente, o quarto e último grupo, Interface Externa, está engajado nas interações entre sistemas baseados em conhecimento e entre outros módulos dentro de um ambiente. A linguagem de comunicação entre agentes provavelmente mais conhecida, KQML (*Knowledge Query Manipulation Language*), surgiu a partir dos trabalhos deste último grupo.

Uma breve descrição sobre KIF, Ontologia e KQML é apresentada nas próximas subseções.

3.4.1 KIF

KIF é uma linguagem formal que serve para se fazer trocas de conhecimento entre programas de computadores escritos por diferentes programadores e linguagens de programação. Embora seja possível de se fazer, KIF não foi projetada para interagir com humanos nem para ter uma representação interna do conhecimento dentro de programas. Tipicamente, quando um programa lê uma base de conhecimento em KIF, ele converte os dados em suas formas internas (e.g. ponteiros). E quando o programa precisa comunicar com outro programa, ele mapeia suas estruturas internas em KIF (Genesereth and Fikes, 1992).

As sentenças em KIF podem ser simples (e.g. tuplas), complexas (e.g. relações) e lógicas (e.g. regras). Pode-se codificar “meta-conhecimento” que representa o conhecimento sobre o próprio conhecimento. A linguagem permite a representação de procedimentos permitindo a construção de estruturas procedimentais.

3.4.2 Ontologias

No contexto de compartilhamento de conhecimento, o termo ontologia significa uma representação de uma especificação, ou seja, é uma descrição formal de conceitos e relacionamentos que são usados por um único agente ou comunidades de agentes.

As ontologias são construídas com o propósito de possibilitar o compartilhamento e a reutilização de conhecimento. Uma ontologia é vista como uma especificação usada para promover compromissos ontológicos entre uma comunidade de agentes. Estes compromissos

sos são descritos utilizando-se ontologias comuns, que definem um vocabulário com o qual consultas e fatos podem ser trocados entre os agentes. Logo, os compromissos ontológicos podem ser vistos como acordos feitos entre os agentes para se utilizar um vocabulário compartilhado de forma coerente e consistente. Dessa forma, os agentes podem comunicar-se sobre um determinado subdomínio sem a necessidade de compartilhar a teoria completa do domínio.

3.4.3 KQML

KQML é uma linguagem (e um protocolo) utilizada para apoiar a comunicação de alto nível entre agentes de software. Ela pode ser usada como uma linguagem por uma aplicação para interagir com vários agentes ou por agentes para interagir cooperativamente na resolução de problemas. KQML não é definida apenas com sintaxe e semântica, ela inclui um protocolo que governa o uso da linguagem (Finin et al., 1997).

Algumas características chaves podem ser resumidas como:

- As mensagens em KQML não comunicam apenas sentenças em uma linguagem, mas elas informam uma atitude a ser tomada sobre o conteúdo da mensagem (e.g. consulta, requisição).
- As primitivas da linguagem são chamadas “performativas”. Como o termo sugere, o conceito destas primitivas está relacionado à ação dos agentes. Estas primitivas definem as ações permitidas que um agente pode fazer durante a comunicação com outros agentes.
- Um ambiente de agentes KQML pode ser enriquecido com agentes especiais, chamados facilitadores, que provêm funções como: associação de endereços físicos com nomes simbólicos; registro de base de dados e serviços de comunicação (e.g. roteamento).
- KQML foi desenvolvida para ser uma linguagem universal de comunicação entre agentes de software, diferentemente de outras linguagens. Por exemplo, os agentes construídos pela linguagem *Telescript* (Seção 3.3.2) possuem dificuldades para comunicar-se com outros tipos de agentes e aplicações que não foram desenvolvidos nesta linguagem.

3.5 Desafios para a Construção de Agentes

As soluções baseadas em agentes têm sido desenvolvidas para diferentes domínios de aplicação, como na indústria (e.g. controle de processos), comércio (e.g. comércio eletrônico), militares, entre outras. E possivelmente não existem restrições quanto às possíveis aplicações desta nova tecnologia. Contudo, apesar dos avanços significativos obtidos na área de AS, Wooldridge (Wooldridge and Jennings, 1998) afirma que pouca importância está sendo dada ao processo de engenharia dos agentes de software.

O receio do autor é que esta falha na engenharia de sistemas acabe frustrando as expectativas dos pesquisadores no desenvolvimento de sistemas práticos, confiáveis, úteis e inteligentes, como aconteceu com os Sistemas Especialistas e as de Redes Neurais. Assim, ele faz uma seleção de algumas categorias problemáticas para o desenvolvimento de sistemas de agentes:

- *Político*: os agentes irão e estão ajudando na construção de software (engenharia), porém, eles não devem ser vistos como elementos mágicos que podem resolver tudo. Outro ponto relevante ao superestimar as capacidades dos agentes é o fato de que muitos projetistas imaginam que os agentes podem raciocinar e agir iguais aos seres humanos. Isto pode criar uma expectativa falsa sobre as habilidades dos agentes.
- *Gerenciamento*: é comum acontecer de uma nova tecnologia consolidar-se e a partir daí ser gerada tanta “propaganda” sobre o assunto que várias empresas incentivadas pela oportunidade de lucro começam a imaginar possíveis soluções utilizando agentes de software sem ao menos estudar as verdadeiras vantagens dos agentes sobre as tecnologias existentes ou sem saber para quais aplicações os agentes realmente são adequados.
- *Conceitual*: devido ao fato da idéia geral dos agentes ser intuitiva, eles estão tornando-se muito “populares”. Este fato é bom por um lado, pois, a disseminação desta tecnologia em diversas disciplinas pode promover a aplicação de agentes através de experimentos em diversas áreas de atuação. Por outro lado, alguns projetistas ficam encorajados a acreditar que dominam todos os conceitos sobre AS e, na realidade, podem estar confundindo os mesmos.
- *Análise e Projeto*: normalmente, quando se está construindo um sistema de agentes, a porção deste sistema envolvida especificamente com a tecnologia de AS pode ser muito

inferior a quantidade dos conceitos existentes das tecnologias tradicionais. Dessa forma, ao invés de se tentar “reinventar a roda”, os projetistas devem analisar as possíveis soluções já existentes e usá-las quando parecer adequado, como CORBA, Sistemas Reativos e Sistemas Especialistas.

- *Nível Micro (agente)*: no projeto de um sistema de agentes, muitos projetistas ficam tentados em criar sua própria arquitetura de agentes ignorando outras existentes que possuam anos de desenvolvimento e dezenas de pesquisadores envolvidos. Um outro problema, é a utilização excessiva de conceitos de IA em sistemas de agentes, tornando-os muito complexos de serem implementados, ou por outro lado, a falta de utilização desses conceitos pode levar a classificações errôneas de programas “comuns” como sendo agentes.
- *Nível Macro (sociedade)*: os projetistas começam a acreditar que podem aplicar agentes a todos os problemas, isto pode gerar descrédito para a área, uma vez que nem todos os problemas podem ser resolvidos usando agentes. Outro problema diz respeito ao número de indivíduos existentes em uma sociedade de agentes, que pode ser muito elevado tornando um sistema incontrolável, e quando é muito reduzido pode-se sobrecarregar tarefas para poucos agentes que poderiam estar sendo feitas paralelamente por mais agentes.

3.6 Considerações Finais

Neste capítulo foi apresentada uma visão geral de alguns aspectos que englobam a área de Agentes de Software como, a tipologia dos agentes e de seus ambientes, projetos de agentes e, por fim, a comunicação dos agentes.

Sistemas de agentes fornecem novas alternativas ao desenvolvimento de programas de computador melhorando suas qualidades e facilitando a sua programação. Ainda, foi apresentado que com a aplicação de alguns conceitos de Inteligência Artificial torna-se possível a construção de sistemas que possuam um certo grau de inteligência e autonomia que possam auxiliar os seres humanos em diversas áreas do conhecimento.

A partir de todos estes conceitos apresentados neste capítulo, este trabalho, utiliza alguns deles para propor o projeto de agentes de software capazes de auxiliar a realização

do processo de Extração de Conhecimento de Base de Dados. Como citado anteriormente, foram consideradas, neste trabalho, as seguintes propriedades para definir um agente de software: autonomia, mobilidade, reatividade, pró-atividade e persistência.

Capítulo 4

Proposta do Sistema Minador

O processo de Extração de Conhecimento de Base de Dados (Capítulo 2), é composto por diversas etapas como: seleção de dados, processamento de dados e *Data Mining*. Várias ferramentas podem ser utilizadas para a execução de cada uma dessas etapas, levando os usuários a conhecer os detalhes dos softwares usados para realizar a extração de conhecimento.

Possivelmente, num grande projeto de KDD, diversos profissionais estarão envolvidos e cada um deverá operar uma ou mais ferramentas específicas. Esta abordagem dificulta a integração dos trabalhos, pois, no momento que os usuários precisam transferir os resultados obtidos, pode-se descobrir que muitas das ferramentas não interoperam automaticamente, sendo necessário tratar os resultados fornecidos para torná-los “integrados”.

Ferramentas especializadas em executar determinadas tarefas sempre irão existir, por isso, é aparentemente improvável que uma organização venha fazer uso de um único software para extrair conhecimentos dos seus dados. Logo, a interoperabilidade destes softwares sempre será um problema. Para tentar solucionar, ou amenizar, essa dificuldade, pode-se utilizar sistemas ou ambientes que sirvam como uma ponte entre os aplicativos usados, tratando suas entradas/saídas e integrando, de certo modo, as ferramentas que dão suporte ao processo de Extração de Conhecimento de Base de Dados.

Ao se fazer uma “integração” dessas ferramentas, torna-se factível, por exemplo, gerenciar diferentes instâncias do processo de KDD, facilitar a troca de informações, melhorar a comunicação entre os usuários, manter históricos de processos realizados, ou seja, garantir uma boa administração do processo. Tudo isto, sem dúvida, aumenta as chances de se obter

resultados melhores e em menos tempo, algo desejado por todas as organizações.

Nesse ínterim, é apresentada a proposta de um sistema para apoiar a execução do processo de KDD. O sistema proposto é composto por agentes de software que possuem algumas das características apresentadas no Capítulo 3. Estes agentes executam tarefas definidas pelos usuários auxiliando a realização do processo de extração de conhecimento. A próxima seção apresenta com mais detalhes os motivos que incentivaram a proposta do sistema.

4.1 Motivação

Normalmente, diferentes ferramentas são usadas em cada etapa do processo de descoberta de conhecimento. Pode-se escolher livremente as ferramentas mais adequadas para se fazer algum tipo de tarefa, ou mesmo, aquelas com as quais a equipe envolvida possui maior conforto e experiência em usar. Porém, tais aplicativos podem não interoperar adequadamente, tornando-se difícil, ou inviável, a comunicação direta entre os mesmos. Os analistas devem portanto realizar esta comunicação através de ajustes e programação em cada ferramenta, sempre que for necessário e possível.

Tentar adequar as saídas de uma etapa como sendo a entrada da próxima pode exigir grande esforço e tempo dos envolvidos no processo de KDD. Além disso, uma equipe de analistas deveria ficar dedicada a este tipo de atividade, o que faria com que os mesmos fossem desviados de sua real atribuição, a descoberta de padrões embutidos nos dados. Tudo isto pode vir a diminuir a produtividade, gerando resultados pobres e de pouco valor para as organizações.

Por se tratar de um processo altamente iterativo e interativo, o gerenciamento do processo de KDD também é um ponto que merece atenção especial. As atividades pertencentes a cada etapa precisam, geralmente, ser repetidas diversas vezes antes de serem aprovadas e concluídas. As iterações são necessárias devido à grande diversidade de técnicas e algoritmos possíveis de ser empregados para se executar uma mesma tarefa e, por isso, diversas combinações precisam ser comparadas a fim de se selecionar a melhor delas. Dessa forma, caso não for usada nenhuma ferramenta para organizar, armazenar e distribuir os “sub-resultados” das etapas de KDD, o seu gerenciamento pode tornar-se difícil e trabalhoso. Isso aumenta a incidência de perguntas do tipo “Quais foram as restrições usadas para coletar dados da

tabela Clientes?”, “Quais *features* o especialista disse que poderiam ser agregados?”, “Quais foram os parâmetros usados para o algoritmo de classificação X?”.

Os softwares conhecidos como *suites* de mineração, são ferramentas capazes de apoiar a execução de algumas ou de todas as etapas do processo de KDD. Os exemplos mais conhecidos são: SGI MineSet ¹, DMSK ², SAS Enterprise Miner ³, IBM Intelligent Miner ⁴ e Clementine ⁵. Com esses *suites* pode-se coletar, processar e minerar os dados de um repositório ou mesmo de um simples arquivo do tipo *flat*. Entretanto, como sempre ocorre, não existe nenhum que possa ser considerado o melhor de todos, uma vez que cada qual possui suas características peculiares. Assim, uma mesma organização pode utilizar diferentes soluções para tarefas distintas, e em muitos casos, o problema da integração e gerenciamento dessas ferramentas está de volta.

A velocidade com que uma organização consegue identificar e solucionar problemas nos dias de hoje, pode ser a diferença entre o sucesso e o fracasso da mesma num mercado extremamente competitivo e globalizado. Até organizações governamentais e de pesquisa podem ser incluídas nesse ambiente. Um exemplo recente pôde ser visto mundialmente, o da corrida pelo mapeamento do DNA Humano (Projeto Genoma). Assim, é necessário que o tempo e os recursos gastos para se fazer o processo de extração de conhecimento sejam reduzidos.

Um dos maiores desafios que uma organização, que pretende utilizar o KDD, pode enfrentar, é assegurar que suas técnicas de análise e processamento são usadas apropriada e corretamente e que as atividades das equipes sejam perfeitamente controladas e coordenadas. A aplicabilidade das técnicas pode depender de um número de fatores, incluindo a questão a ser tratada, as características dos dados sendo estudados e a história do processamento destes dados.

Os problemas de gerenciamento do processo de KDD podem agravar-se, caso a equipe de analistas não possua experiência com as técnicas usadas ou se existir muita diferença no nível de conhecimento entre os analistas com relação aos dados e familiaridade com uma técnica de análise em particular. Assim, por exemplo, um grupo de analistas que domina

¹<http://mineset.sgi.com>

²<http://www.data-miner.com/>

³<http://www.sas.com/software/components/miner.html>

⁴<http://www.ibm.com/software/data/iminer/fordata/>

⁵<http://www.spss.com/clementine/>

alguma técnica específica de análise, tenderá a usar sempre a mesma técnica.

O número de pessoas envolvidas num projeto de extração de conhecimento e a posição geográfica das mesmas é uma outra dificuldade no gerenciamento do processo de KDD, pois todos os usuários devem conseguir acessar os resultados obtidos de cada etapa em tempo hábil, independentemente do horário e local onde estejam. Além disso, a troca de informações entre eles deve ser feita de forma a agilizar e melhorar a validação dos resultados conseguidos.

Grande parte das tarefas pertencentes às etapas do processo de KDD é repetitiva, trabalhosa e, algumas vezes, tediosa. Por exemplo, para coletar dados de um SGBD (Sistema Gerenciador de Base de Dados), um usuário quase sempre escreve consultas SQL, conecta-se a base de dados e aguarda os resultados da execução das mesmas. Essas consultas podem levar minutos ou horas para serem processadas, dependendo da quantidade de dados existente. Além disso, esta tarefa pode vir a ser repetida dezenas de vezes. Dessa forma, um usuário fica praticamente preso a uma tarefa cuja única parte intelectual foi escrever as consultas na linguagem SQL. Conseqüentemente, o trabalho restante deve ser feito automaticamente, por um software autônomo capaz de conectar a base, rodar as consultas, recuperar os resultados e enviá-los diretamente para o usuário, que, por sua vez, estaria focado em outras atividades dentro do processo.

Uma possível solução para facilitar e melhorar os resultados obtidos no processo de KDD pode ser alcançado fazendo uso de agentes de software (Uthurusamy, 1996). Os agentes são capazes de executar algumas tarefas, como: coletar dados assim que estes estejam disponíveis, realizar o pré-processamento seguindo especificações prévias de usuários, tomar decisões simples baseadas num conhecimento anterior e apresentar os resultados para todos os envolvidos no processo. A seguir são citados alguns trabalhos que utilizam agentes para auxiliar a execução do processo de KDD.

4.2 KDD Apoiado por Agentes de Software

O uso dos chamados Agentes de Software para apoiar o processo de KDD vem sendo pesquisado por algumas instituições de pesquisas e empresas. Alguns destes trabalhos tratam da recuperação de informação de fontes heterogêneas (Oates et al., 1994; Knoblock et al., 1994; Knoblock and Ambite, 1997), integração e tratamento de dados distribuídos (Ashish et al., 1997; Ambite et al., 1998), mineração de dados distribuída (van de Velde, 1996; Kargupta

et al., 1997; Kargupta et al., 1999) e o processo de KDD num ambiente distribuído (Davies and Edwards, 1995; Stolfo et al., 1997; Yang et al., 1998; Unruh et al., 1998; Jensen et al., 1999).

Dois projetos específicos foram estudados com maior profundidade: o Little-JIL (Jensen et al., 1999) e o InfoSleuth (Bayardo, 1997; Unruh et al., 1998; Nodine et al., 1998). A razão para se utilizar estes projetos foi o fato de que ambos enfatizam a utilização de agentes em praticamente todo o processo de KDD, procurando tratar desde a coleta de dados até a validação do conhecimento obtido de formas distintas. As duas abordagens serviram como uma base inicial para o projeto da dissertação.

Nas próximas subseções estão apresentadas uma breve descrição de ambos.

4.2.1 Little-JIL

Baseados na programação de processos para solucionar problemas de coordenação no desenvolvimento de software (engenharia de software), os pesquisadores envolvidos neste projeto utilizaram as mesmas técnicas para representar e apoiar a coordenação no processo de KDD.

Uma especificação de processo define uma classe de instância de processo. Cada instância segue a especificação, entretanto a execução das mesmas difere de acordo com os agentes e dados disponíveis quando o processo é executado. As instâncias diferem umas das outras de diferentes formas que incluem a seleção dos agentes que executam tarefas particulares, a ordem na qual as tarefas são executadas e na escolha das subtarefas usadas para completar uma dada tarefa.

Como o trabalho de capturar e representar explicitamente de forma precisa, completa e clara os processos não é um trabalho trivial, foi utilizado a linguagem de especificação de processos Little-JIL para facilitar um pouco esta tarefa. Dessa forma, é explorado como coordenar as atividades de múltiplos agentes de KDD. Estes agentes representam todo e qualquer tipo de agente que interaja com o sistema seja ele um software ou uma pessoa.

No projeto Little-JIL não existe a preocupação de se programar agentes de software individuais para executar tarefas como treinar uma rede neural ou calcular estatísticas. Tais tarefas são melhor implementadas utilizando-se linguagens de programação convencionais como C++ ou Java. Além disso, não é feita nenhuma tentativa de ensinar como um analista deve executar suas tarefas, pois o mesmo possui experiência e conhecimento. A tentativa é

explorar linguagens flexíveis que possam ser usadas para coordenar as ações dos analistas humanos experientes com os agentes de software para se construir o processo que permita aos analistas menos experientes atingir melhores resultados.

4.2.2 InfoSleuth

No projeto InfoSleuth o foco é suprir as necessidades de clientes por tecnologias que forneçam soluções como: tratar com informação em diferentes níveis de abstração e em vários formatos de mídias; fundir informações sobrepostas vindas de múltiplas fontes em um único local integrado, e monitorar e reagir às mudanças, ou padrões de mudanças ocorrendo nas fontes de informações distribuídas pela rede.

InfoSleuth é um sistema baseado em agentes usado para executar tarefas de recuperação, integração e análise através de redes de fontes de informações. Tal sistema é definido como um conjunto de processos interoperáveis, cooperativos e ativos distribuídos através de uma rede ou Internet. Cada agente é especialista em uma tarefa específica, múltiplos agentes interagem e cooperam a fim de solucionar tarefas de integração e análise.

Todos os agentes do sistema são construídos em cima do *InfoSleuth Agent Shell*, que é escrito em Java. Os agentes comunicam-se utilizando KQML (Finin et al., 1997). O sistema possui diversos tipos de agentes como agentes de consultas de recurso e de multi-recursos, agentes sentinela e de mineração. Um exemplo de aplicação é a aquisição, integração e monitoramento de informações técnicas de Inteligência Competitiva a partir de fontes abertas.

A atividade primária para um problema de Inteligência Competitiva é correlacionar informações das fontes, encontrar tendências e associações entre as fontes e detectar variações de tendências através do tempo. Para tanto, resumidamente, são utilizados agentes de consultas de recursos para trazer informação no sistema e realizar a integração entre diferentes fontes. Os agentes de mineração e sentinela permitem que os usuários adicionem informações sobre a maneira como os dados estão correlacionados e modificados. Os agentes sentinelas podem disponibilizar tarefas de *Data Mining* ajustando um “workflow” para enviar instruções algorítmicas e de dados para os agentes de mineração. Os agentes de mineração executam tarefas do processo de KDD, como seleção de dados relevantes, pré-processamento, aplicam algoritmos de análise e interpretam os resultados.

4.2.3 Considerações sobre os Projetos InfoSleuth e Little-JIL

Projetos como o InfoSleuth e Little-JIL já estão bem avançados na aplicação de agentes de software para executar coleta de dados, pré-processamento e descoberta de conhecimento. Como foi visto, o InfoSleuth é um sistema de agentes distribuídos de coleta e análise de dados, apresentando vários agentes que realizam tarefas complexas de KDD. Little-JIL é uma linguagem para se descrever um processo para coordenar a execução do processo de KDD usando agentes, sejam humanos ou de software. O primeiro procura enfatizar as propriedades dos agentes para executar tarefas de coleta e descobrimento. O segundo ressalta mais o lado da coordenação, fluxo de controle e definição dos pré e pós-requisitos do processo.

Algumas das características existentes em cada um dos dois projetos citados foram utilizadas como requisitos iniciais ou idéias para a elaboração do sistema multi-agente Minador para apoiar o processo de Extração de Conhecimento de Base de Dados. Por exemplo, pode ser observado no sistema Little-JIL que é interessante permitir a definição das pré e pós-condições de um processo de KDD bem como usar o conhecimento do analista para permitir que os agentes executem suas tarefas. Os agentes podem ser subdivididos em tipos, como no InfoSleuth, capazes de executar tarefas particulares de acordo com uma especialidade podendo acarretar numa redução de complexidade no projeto e na implementação dos agentes do sistema. Assim, estas características entre outras foram usadas como elementos iniciais para se propor a elaboração do sistema Minador.

A visão geral da proposta deste sistema e suas características peculiares estão apresentadas nas próximas seções.

4.3 Visão Geral do Sistema Minador

A elaboração do sistema Minador visa servir como uma nova base de idéia para a implementação de sistemas baseados em agentes de software, ou sistemas multiagentes, que possam ser aplicados na Extração de Conhecimento de Base de Dados.

Algumas das idéias dos projetos InfoSleuth e Little-JIL (Seção 4.2) foram usados na formação das idéias contidas neste trabalho. O propósito geral do sistema Minador é muito semelhante ao apresentado pelos dois projetos citados que é a utilização de agentes para a realização do processo de KDD. Porém, algumas finalidades e propriedades, apresentadas

nas próximas seções, do sistema o faz ser um projeto único neste genero.

4.3.1 Semelhanças entre Trabalhos Relacionados

Os dois projetos previamente referenciados trabalham com o termo instância, que para o Minador representa um processo, com pré e pós-requisitos estabelecidos, sendo executado. Ou seja, podem existir diversas instâncias sendo trabalhadas pelos mesmos ou diferentes agentes do sistema. Os valores de pré-condição definem os parâmetros necessários para que uma instância seja iniciada e os de pós-condição definem os requisitos de finalização de um instância do processo de KDD.

Seguindo a idéia do sistema Little-JIL, um processo deve ser estabelecido de forma explícita utilizando-se o conhecimento do analista, ou seja, o sistema é capaz de obter instruções mais complexas (intelectualmente) do analista e utiliza os agentes para executá-las. Para tanto, os agentes devem ser capazes de iniciar autonomamente ações auxiliares necessárias a fim de executar tarefas fundamentais. Para melhor ilustrar tal situação, pode-se tomar como sendo uma tarefa fundamental recuperar dados segundo um critério (que o analista precisa inserir na tarefa), o agente pode então perceber que o analista alterou ou criou uma nova tarefa e em seguida realizar a conexão na base de dados e executar a tarefa do usuário, guardando o resultado obtido. Todas estas tarefas que o agente executa autonomamente são tarefas auxiliares.

Do projeto InfoSleuth algumas idéias como a recuperação de dados de fontes heterogêneas e camada de abstração permitiram a formação de uma tipologia de agentes no sistema Minador. Ou seja, são usados agentes em diferentes funcionalidades de acordo com as características peculiares de cada agente do Minador. Os agentes comunicam-se trocando mensagens e respondendo a eventos gerados do sistema. A recuperação e o tratamento de informações foi elaborado para ocorrer de forma semelhante, sendo que existem agentes específicos para unir e integrar dados disponibilizando-os para os próximos agentes existentes no sistema. Ainda, os objetos Eventos existentes no sistema InfoSleuth contribuíram para a criação do esquema do disparo, percepção e reação aos eventos tratáveis do sistema.

4.3.2 Diferenças entre Trabalhos Relacionados

O foco central do sistema Minador é a integração de diferentes ferramentas para auxiliar o gerenciamento do processo de KDD através de agentes autônomos. Nesse ínterim, ele apresenta uma estrutura mais simplificada que a mostrada pelo sistema InfoSleuth. Além disso, seus agentes realizam atividades pouco complexas e são dependentes de outras ferramentas para executar suas tarefas. Conseqüentemente, o número de agentes é menor e eles podem executar mais tarefas uma vez que eles não precisam implementá-las, mas simplesmente usar as funções das ferramentas. Em contra partida, a complexidade para se fazer uma interface unificada que utilize os recursos de ferramentas distintas pode ser grande.

Não é utilizada nenhuma linguagem de processo, como no Little-JIL, para se construir os agentes, assim a coordenação do processo não é determinada pela linguagem, mas sim pelos usuários que devem controlar todo o andamento do mesmo. Isto pode tornar o gerenciamento mais difícil, contudo, pode garantir uma maior flexibilidade para os analistas.

4.3.3 Contribuições Oferecidas

O sistema não opera com processos muito complexos, onde a aprendizagem é o ponto forte, mas sim, foi elaborado para ser um sistema que esteja apto a realizar tarefas simples e rotineiras para servir como um assistente ao usuário especialista, permitindo que o mesmo aplique seu conhecimento e experiência em tarefas mais intelectuais que “braçais”. Para tanto, são usados agentes assistentes para executar tarefas que possam ser feitas autonomamente pelos agentes, como: conectar a um repositório de dados; coletar amostras aleatórias de dados; selecionar *features*; criar arquivos de dados com formato específico, os quais são usados por algoritmos de mineração de dados; distribuir os resultados obtidos via páginas de *Web*. Os agentes devem ser capazes de interoperar com as ferramentas usadas na execução das tarefas do processo de KDD.

A execução do sistema ocorre de forma distribuída, ou seja, é feita em diversos computadores interligados via rede. Essas máquinas devem possuir ambientes de execução, chamados de Ambientes de Execução de Agentes (AEA), os quais possibilitam aos agentes trafegar livremente por entre os ambientes. Cada AEA cumpre papéis específicos, como: servidor de dados, servidor de mineração e servidor de *Web*. É importante observar que esta configuração não é rígida. Assim, um único ambiente pode acumular duas ou mais funções,

além de ser possível existir mais de um desempenhando o mesmo papel no sistema. Um servidor de ambiente auxilia os AEAs a localizar outros ambientes e agentes existentes no Minador, além de gerenciar todas as trocas de mensagens e objetos entre eles.

Os agentes existentes no sistema executam tarefas repetitivas e simples que não precisam de constante supervisão por parte de um analista, como, por exemplo, a coleta de dados. Assim, ao serem especificados quais tabelas e campos serão coletados, o sistema envia ordens para que um agente faça a conexão e amostragem dos dados nas tabelas. Caso não exista nenhum agente disponível, um novo pode ser criado automaticamente.

O Minador possui diferentes tipos de agentes especializados em executar tarefas pertencentes a algumas das etapas do processo de KDD. Estes agentes são classificados de acordo com suas funcionalidades e habilidades. Porém, todos herdam as características apresentadas na Seção 3.1 (Capítulo 3) de autonomia, reatividade, pró-atividade, persistência e mobilidade de um único tipo de agente base. A partir destas propriedades, são adicionadas outras específicas para a realização de determinada tarefa, caracterizando novos tipos de agentes.

Diversos experimentos e testes foram feitos visando verificar a validade de algumas das idéias, como por exemplo, AEAs independentes de plataforma, recuperação automática de dados, mobilidade, entre outras. Foram implementadas algumas das funcionalidades propostas no sistema necessárias para indicar a viabilidade de se utilizar determinada técnica ou ferramenta. Os softwares utilizados para a realização dos testes foram o SGBD Oracle8i(tm) ⁶, o *suite* de mineração MineSet(tm) e o servidor de Web Apache(tm) ⁷. Para a implementação dos testes, está sendo utilizada a linguagem de programação Java(tm) ⁸ e a API Jumping Beans ⁹, para a construção dos ambientes de execução dos agentes.

Os detalhes da proposta do sistema Minador envolvendo sua arquitetura, seus agentes, os ambientes e módulos de controle estão apresentados nas seções seguintes.

⁶<http://www.oracle.com>

⁷<http://www.apache.org>

⁸<http://java.sun.com>

⁹<http://www.JumpingBeans.com>

4.4 Arquitetura do Minador

O sistema Minador está organizado em uma estrutura dividida em camadas. As camadas representam diferentes componentes que formam o sistema. Como pode ser observado na Figura 4.1, os componentes existentes são: Interface de *Web*, Gerenciador de Descobrimento, Agentes e os Ambientes de Execução de Agentes (AEA). Além destes componentes nas duas últimas camadas ficam as Máquinas Virtuais Java ou *Java Virtual Machine* (JVM) e a rede de computadores. Estes componentes possibilitam, respectivamente, a independência de plataforma e mobilidade dos agentes.

Figura 4.1: Arquitetura do Sistema Minador.

Na primeira camada, Interface de *Web*, ficam localizados os componentes de interface gráfica do usuário. Através desta interface são feitas todas e quaisquer interações entre os usuários e os demais componentes do sistema.

Na segunda camada, encontra-se o componente central do sistema, o Gerenciador de Descobrimento, pelo qual os usuários podem monitorar e administrar a execução de instâncias do processo de KDD, bem como todas as demais operações do sistema.

A camada Agentes é constituída pelos agentes de software. Estes componentes exe-

cutam as tarefas que os usuários requisitam através do Gerenciador de Descobrimento via Interface de *Web*. Os agentes encaminham os resultados de volta aos usuários utilizando o caminho inverso.

Na camada seguinte estão os Ambientes de Execução de Agentes. Os AEAs são os locais onde os agentes executam todas as suas atividades, deslocando-se livremente por entre os ambientes. Os AEAs rodam em computadores que possuem recursos (e.g. SGBD) utilizados pelos agentes para realizar suas tarefas.

Os componentes JVM e Rede de Computador são considerados como infra-estrutura para a criação da arquitetura do sistema. A camada de JVM apresenta Máquinas Virtuais Java, que são sistemas prontos disponíveis gratuitamente para serem utilizados. Está camada está presente pois os agentes e AEAs são implementados em Java e precisam das mesma para executar. Cada AEA executa em sua própria máquina virtual, já os agentes podem estar rodando em qualquer uma. As Máquinas Virtuais Java são executadas em computadores distribuídos numa rede de computadores representada pela última camada da arquitetura. Ressalta-se que as redes podem ser tanto locais (LAN - *Local Area Network*) quanto globais (WAN - *Wide Area Network*).

Nas próximas seções são apresentadas, em mais detalhes, as funcionalidades da Interface de *Web*, Gerenciador de Descobrimento, Agentes e AEA.

4.5 Interface de *Web*

Em um projeto de KDD, normalmente, existem três tipos de usuários (Seção 2.3): usuário final, o especialista do domínio e o analista. Cada qual com necessidades e papéis específicos. Além disso, é possível que os usuários estejam trabalhando em ambientes diferentes dentro de uma empresa, ou até em localidades (e.g. estados) diferentes. Por isso, faz-se necessário disponibilizar o sistema Minador de forma mais abrangente possível.

Um fator importante a se considerar é a flexibilidade com que os usuários acessam o sistema. Com o advento da Internet, as pessoas conseguem comunicar-se umas com as outras por diversos artifícios na rede, e o mais utilizado e mais difundido por todo o mundo é a *World Wide Web*, ou simplesmente, *Web*. Através dela podemos acessar em tempo real informações nos mais diversos formatos, como: figuras, textos, filmes e sons.

Com a idéia de facilitar o acesso ao sistema Minador independentemente de hard-

Figura 4.2: Interface de *Web* do Sistema Minador.

ware, sistema operacional e local físico, os usuários interagem com o sistema de qualquer localidade no mundo através de uma Interface de *Web* (Figura 4.2). Esta interface possui uma arquitetura cliente/servidor *3-tier*, ou seja, para acessar o sistema, é preciso fazer uso de *Browsers* (cliente) de navegação, capaz de comunicar-se com um Servidor de *Web* (servidor), que por sua vez, comunica-se com o Minador (servidor). O caminho desta comunicação é feito nos dois sentidos entre cada par de componentes.

4.5.1 Interface Gráfica do Usuário

Basicamente, a interface gráfica disponível ao usuário é composta por páginas HTML (*Hyper Text Mark-up Language*). Estas páginas provêm um ambiente amigável ao usuário, de onde é possível interagir com o sistema. Além disso, com o intuito de satisfazer as necessidades de cada usuário, a mesma deve possuir uma arquitetura dinâmica, garantindo um alto grau de interatividade entre o usuário e o sistema.

Para dar dinamismo às páginas da interface são utilizados, no lado cliente, *Java Applets*, *Scripts* e *Java Server Pages* (JSP). O primeiro possui a característica única de ser enviado até o *Browser* do usuário para executar algum tipo de processamento (e.g. animação). Os *Scripts* são programas interpretados inseridos no meio das páginas HTML, dando “vida” às páginas. As JSP, são páginas HTML contendo código Java criadas dinamicamente,

que podem atuar em conjunto com um *Servlet*. No lado servidor, existem *Servlets* ou CGI (*Common Gateway Interface*), que processam os pedidos feitos pelos usuários repassando-os ao sistema Minador.

É necessário que um Servidor de *Web* esteja disponível e rodando para acessar o sistema Minador. Este servidor pode ou não estar instalado no mesmo computador onde o Gerenciador de Descobrimento (Seção 4.6) executa. Entretanto, o desempenho da interface em termos de velocidade está ligado diretamente a esta configuração. Assim, se os dois estiverem instalados numa mesma máquina o desempenho do sistema poderá ser melhor, pois existirá menos tráfego de dados via rede.

4.5.2 Funcionamento da Interface

Um esquema do funcionamento da Interface de *Web* pode ser visto na Figura 4.2. Todos os serviços disponíveis podem ser requisitados através de *browsers*, que, por sua vez, enviam os pedidos para o servidor de *Web*. Assim, dependendo do serviço pedido, o servidor pode chamar a execução de um *Servlet* que passa a requisição ao gerenciador de descobrimento. Os resultados podem ser enviados aos *browsers* via páginas JSP.

Dentro deste esquema, a Interface de *Web* possui como componente central o Servidor de Aplicações que faz a interface entre o Minador e o servidor de *Web*. Este componente deve estar conectado ao servidor de *Web* utilizado. Assim, o usuário tem condições de solicitar serviços para o Gerenciador de Descobrimento, como: iniciar os AEAs, monitorar o andamento das instâncias do processo, visualizar resultados, entre outros.

Como a interface do sistema está disponível via Internet, os usuários podem acessar o Minador de qualquer lugar, não precisando estar fisicamente nos locais onde ficam as máquinas pertencentes ao sistema.

4.6 Gerenciador de Descobrimento

Por intermédio do Gerenciador de Descobrimento é possível administrar as instâncias que estão sendo executadas no Minador. Os usuários interagem com o gerenciador via a Interface de *Web*. Assim, pode-se requisitar serviços aos agentes do sistema, como: coleta de amostras, transformação dos dados, preparação de arquivos de dados para os algoritmos/ferramentas

de mineração e distribuição dos resultados.

Figura 4.3: Organização dos Módulos do Gerenciador de Descobrimento.

O gerenciador está organizado em módulos de controle, como pode ser visto na Figura 4.3. Através destes módulos, pode-se configurar e monitorar as instâncias. O gerenciamento das etapas do processo é feito utilizando-se os módulos de processamento de dados, mineração de dados e de distribuição. O núcleo do sistema foi projetado de forma a permitir que novos módulos possam ser acrescentados ao sistema, aumentando, assim, suas funcionalidades.

A comunicação com os agentes é feita indiretamente via o gerenciador, que envia mensagens para os mesmos. O monitoramento dos agentes é feito verificando-se continuamente o status de cada um, o local onde se encontra e o histórico de operações feitas. As próximas subseções mostram os cinco módulos do gerenciador em detalhes.

4.6.1 Módulo de Configuração de Instâncias

Para que um processo de KDD possa ser iniciado, o usuário deve configurar a instância a ser executada através do Módulo de Configuração de Instância. Neste módulo é possível criar usuários, descrever o objetivo do processo, atribuir papéis, especificar os repositórios de (*logs*) de execução e configurar os AEAs.

Uma instância de KDD é um processo específico de extração de conhecimento. Para ilustrar melhor este conceito, pode-se pensar numa provável situação em que dois analistas desejam realizar análises sobre os dados das vendas da empresa *X*. Um deles gostaria de analisar dados nas tabelas *Vendas* e *Clientes*, sendo que os campos nulos da tabela *Vendas* deverão ser transformados para o valor zero. O objetivo dele é identificar agrupamentos

(*clusters*) dos consumidores, porém, deseja que o algoritmo utilizado forneça, no máximo, três agrupamentos distintos de clientes. Já o outro analista usará dados das tabelas de *Vendas*, *Cliente* e *Produto* para encontrar regras de associações entre produtos comprados por clientes do sexo masculino, restringindo o valor mínimo da compra a um determinado número. Assim, uma instância representa um processo específico de um usuário com requisitos diferentes.

4.6.2 Módulo Monitor do Sistema

O andamento do processo é monitorado pelo Módulo Monitor do Sistema. Este módulo mostra informações sobre status das tarefas do processo de extração de conhecimento, disponibilidade de recursos e *logs* relativos a uma instância de KDD específica.

A inicialização de uma instância é feita a partir deste módulo. Uma vez tendo sido iniciada, os usuários podem utilizar as funcionalidades existentes para trocar informações sobre as tarefas, escrever pareceres sobre os resultados e enviar mensagens a outros usuários.

4.6.3 Módulo de Processamento de Dados

Neste módulo estão presentes as funcionalidades que permitem o sistema acessar as fontes de dados, especificar como será feita a coleta de dados, bem como a transformação nos mesmos e a criação de bases temporárias.

As fontes de dados, para o Minador, constituem todos os possíveis repositórios de informações, tais como, base de dados, *Data Warehouse*, arquivos ASCII, *sites* de *Web* e bases de conhecimento.

Para se coletar amostras de uma ou mais fontes de dados é necessário que sejam especificados alguns critérios de busca, além de outras informações como o nome do usuário e senha. Os critérios de coleta permitem aos agentes executar consultas SQL sobre dados. Ainda, informações sobre os locais das fontes, as permissões, entre outros, devem ser fornecidas neste módulo.

As transformações feitas sobre a base de amostra coletada podem ser filtragens, agregações, “sumarizações” e até criação de novos campos ou tabelas. Para possibilitar a execução destas operações, o usuário especifica no Módulo de Processamento de Dados, via uma interface visual, os critérios e ações a serem realizadas pelos agentes. Estas informações são usadas para compor sentenças na linguagem SQL. Para os usuários que dominem a sintaxe

da SQL, eles podem editar as sentenças *scripts* diretamente na linguagem.

4.6.4 Módulo de Mineração

Apesar do nome, o objetivo inicial do Módulo de Mineração, não é executar o processo de “Data Mining”, mas sim, especificar os componentes necessários para realizar a mineração de dados utilizando-se alguma ferramenta ou algoritmo. Estes componentes são basicamente, dois arquivos: o arquivo de configuração e o arquivo de dados. Assim, neste módulo o usuário poderá especificar o algoritmo e/ou a ferramenta a ser usada, parametrizar alguns algoritmos, apontar o números de registros a serem usados, escolher os nomes dos atributos, entre outros. A partir destas informações os agentes criam os arquivos necessários que podem ser usados pelos usuários para executar a mineração de dados.

Os arquivos de configuração são dependentes do algoritmo ou da ferramenta utilizada para realizar a mineração de dados. Assim, diversas especificações dos arquivos usados por cada algoritmo devem estar catalogadas. Nesta proposta, para demonstração, serão utilizados as especificações dos arquivos de configuração e de dados do *suite* MineSet.

Os arquivos de dados podem ser divididos em arquivos de treinamento, teste e de validação. Assim, pode-se separar uma determinada amostra em diversos arquivos de dados para os algoritmos a serem utilizados. A intenção é utilizar algumas técnicas bastante difundidas como *holdout* e *resampling* para a criação dos arquivos de dados, estas técnicas estão apresentadas na seção 2.3.5.3.

4.6.5 Módulo de Distribuição

Neste módulo pode ser especificada a “política” de distribuição dos resultados, ou seja, as permissões necessárias, os grupos de usuários que poderão acessar e que serão notificados através de mensagens eletrônicas, o local onde estarão disponíveis todos os resultados obtidos com comentários e opiniões dos envolvidos no processo, entre outros.

O principal objetivo do Módulo de Distribuição é garantir que os resultados obtidos na execução de uma instância estejam disponibilizados ou sejam distribuídos para os usuários envolvidos. Assim, informações sobre as amostras coletadas, as transformações feitas nas mesmas, visualização dos dados, arquivos de regras, árvores de decisão, gráficos e imagens gerados pelos algoritmos de mineração, entre outros, estarão disponíveis pelos canais de

distribuição especificados neste módulo.

Os usuários devem especificar os locais onde irão ficar disponíveis os resultados da mineração feita utilizando-se ferramentas externa de mineração, como o *MineSet*. Dessa forma, os agentes poderão coletar estes arquivos para fazer a distribuição a todos os usuários especificados na política de distribuição.

4.7 Agentes

Os agentes de software do sistema Minador são objetos instanciados de classes Java, que possuem implementadas as seguintes características, vistas na Seção 3.1: reatividade, autonomia, pró-atividade, persistência e mobilidade.

Figura 4.4: Classes básicas de agentes do Minador.

Os agentes de software do sistema Minador podem ser gerados a partir de quatro

classes básicas (descritas mais adiante): Agente Base, Agente de Dados, Agente Minerador e Agente Distribuidor. Além, e a partir destas, outras classes especializadas podem ser criadas para executar tarefas mais específicas. A Figura 4.4 ilustra a hierarquia e algumas das propriedades das classes básicas de agentes de software do Minador.

Todos os agentes derivam de um agente elementar, o Agente Base, herdando, por exemplo, uma máquina de estado, interface de comunicação, gerenciador de segurança e um sistema de ação/reação aos eventos do ambiente. A partir destas propriedades, as classes derivadas implementam outras possibilitando a execução de tarefas mais específicas.

4.7.1 Agente Base

Um Agente Base implementa as características essenciais que os agentes devem possuir para interagir com o sistema Minador. São elas: a Máquina de Estados Finitos, Gerenciador de Segurança, Sistema de Ação/Reação, Memória e a Interface de Comunicação. As propriedades citadas existem independentemente da linguagem escolhida.

A seguir uma visão detalhada das características de reatividade, pró-atividade, mobilidade, entre outras, pertencentes a um Agente Base, é apresentada.

4.7.1.1 Reatividade

Para que os agentes consigam interagir com o seu “mundo”, eles devem ser capazes de responder a determinados eventos ocorridos dentro do mesmo. Para tanto, um agente deve ser capaz de distinguir entre eventos do sistema operacional e os eventos pertinentes ao seu ambiente. Desse modo, por exemplo, eventos disparados de uma criação ou chegada de um novo agente no ambiente deve ser tratado de acordo. Porém, um evento disparado devido a um erro de criação do arquivo não deve ser tratado pois o evento foi gerado pelo sistema operacional.

Esta separação entre eventos ocorridos no ambiente do agente e outros do ambiente computacional faz-se necessária para estabelecer um limite bem definido do que um agente consegue ou não tratar, permitindo a criação de agentes menos complexos e, por isso, mais fáceis de serem implementados.

Os eventos aos quais os agentes devem responder não podem ser genéricos nem ilimitados, ou seja, deve haver um número certo de eventos que estejam dentro do escopo de

execução dos agentes do sistema.

Figura 4.5: Um evento sendo disparado para um agente base.

Cada agente deve possuir uma estrutura interna representando um sensor de eventos capaz de perceber e receber os eventos que estejam ocorrendo no seu mundo, devendo interpretá-los e responder de acordo. Esta estrutura deve ser flexível, permitindo a incorporação de novos tratadores de eventos específicos para cada tipo de agente. Na Figura 4.5 está apresentada uma ilustração de um evento ocorrido e sendo enviado ao sensor de um agente.

Como muitas mensagens podem estar sendo enviadas simultaneamente, muitas vezes, isto ultrapassa a capacidade de recepção e resposta de um agente, por isso, o sensor deve ser capaz de armazenar em um *buffer* as mensagens que ainda não podem ser processadas. Para que novas mensagens importantes não fiquem esperando em uma fila do *buffer* um esquema de prioridade e escalonamento deve ser elaborado.

Uma vez que um evento tenha sido recebido, é necessário interpretar tal mensagem para que a mesma possa ser tratada de acordo. A partir da interpretação, um agente pode responder fazendo alguma ação ou, simplesmente, ignorar o evento (caso não seja de sua responsabilidade tratar o evento).

A arquitetura do sistema Minador possui uma característica distribuída (Seção 4.4). Assim, os eventos tratados por um agente não estão restritos apenas ao ambiente onde ele se encontra, mas sim, a todos os outros ambientes pertencentes sistema Minador. Ou seja, toda e qualquer mensagem pertinente que for disparada no sistema deve ser recebida e tratada por todos os agentes. Na Figura 4.6 pode ser observado que um agente recebe eventos disparados dentro do seu próprio ambiente (*AEA1*), de outros AEAs (*AEA2*) e do sistema.

Figura 4.6: Ilustração do esquema de eventos de um agente base.

4.7.1.2 Pró-Atividade

Reagir aos eventos ocorridos ou às requisições feitas pelos usuários através do sistema garante a um agente a capacidade de reagir e responder prontamente a tais mensagens. Entretanto, o seu fluxo de trabalho não é guiado apenas pelos eventos e requisições, mas muitas vezes, o próprio agente deve ser capaz de tomar a iniciativa e iniciar uma nova atividade ou realizar uma ação. Por exemplo, após finalizar a coleta de dados em um determinado ambiente, um agente pode decidir ficar “acordado” no sistema aguardando uma nova requisição, ou “dormir” no mesmo ambiente, ou “viajar” para outro ambiente e “dormir”, ou se for o caso, “autodestruir-se”.

Implementar a característica de pró-atividade é uma tarefa difícil, pois para um agente tomar uma iniciativa ele precisa considerar diversos fatores e acontecimentos ocorridos, ou que estejam ocorrendo no seu “mundo”, inferindo todas as possíveis consequências. Dessa forma, as habilidades de perceber, planejar e agir, tornam-se extremamente difíceis de ser implementadas.

Como parte do objetivo inicial deste projeto está a construção de agentes com habilidades suficientes para responder uma lista finita de eventos de acordo com regras pré-fixadas. Assim, os agentes do sistema Minador conseguem executar uma lista limitada de atividades relacionadas a alguma tarefa e, uma vez que tenha sido concluída, o agente, normalmente, passa para um novo estado, a partir do qual, existem algumas decisões possíveis de serem tomadas.

Para que um agente resolva que decisão tomar, é feita uma varredura sobre sua Tabela de Decisão interna. Esta tabela apresenta os possíveis estados e os eventos tratáveis específicos de cada agente. As posições da tabela indicam quais ações os agentes podem iniciar. O objetivo é mapear um número suficiente de estados, eventos e decisões para que os agentes consigam operar dentro do seu mundo de acordo com a característica específica de cada agente. Na Tabela 4.1 pode ser observado um exemplo de uma Tabela de Decisão.

Cada decisão tomada dispara uma ação a ser executada pelo agente. Na Tabela 4.1, pode ser visto que uma vez que o agente esteja no estado *Acordado* e este receber uma requisição de iniciar a tarefa (*Inicio Tarefa*), então, uma nova ação é executada, *Trabalhar*.

# Estados	Eventos Disparados				
	Inicio Tarefa	Fim Tarefa	Inicio Ação	Fim Ação	...
<i>Decisões</i>					
Acordado	Trabalhar	Suicidar	-	Próxima Ação	...
Dormindo	Acordar	-	Iniciar Ação	-	...
Trabalhando	-	Dormir n minutos	-	Próxima Ação	...
Trafegando	Ativar	Ativar	-	-	...
....					...

Tabela 4.1: Tabela de Decisão

4.7.1.3 Mobilidade

A mobilidade é a característica que garante que um agente pode executar independentemente do AEA. Isto é muito importante, pois nem sempre os recursos necessários para que um agente trabalhe estão em um único ambiente. E a possibilidade de um agente poder migrar de forma autônoma de um AEA para outro, faz com que o mesmo tenha uma maior flexibilidade para executar diferentes tarefas, continuar uma tarefa, passar informações necessárias a um outro agente localizado no AEA destino, entre outros.

Como cada AEA pode estar executando em diferentes máquinas em uma rede de

computadores, o fato de um agente conseguir transferir-se de um computador hospedeiro (*host*) a outro pode diminuir significativamente o tráfego pela rede, uma vez que todo o processamento de uma tarefa pode ser executado *in loco*. Os resultados de uma determinada atividade pode ser retornada via mensagens enviadas pela rede.

Para que um agente possa locomover-se entre diferentes máquinas em uma rede, é necessário a existência de ambientes que recebam tais objetos, pois, devido a diferenças existentes entre sistemas operacionais, políticas de segurança, privacidade de dados, entre outros, muitas vezes, não é permitido a um software entrar no sistema de uma máquina livremente ou de forma simples. Assim, um ambiente deve ficar executando como um serviço nos computadores prontos para receber agentes em um porto(porta) específico(a).

Os Ambientes de Execução de Agentes ficam recebendo agentes nos diferentes *hosts*. Dessa forma, os agentes executam “debaixo” do processo de execução do AEA, dentro de uma máquina virtual Java. Na Figura 4.7 o agente *AG1* encontra-se no *AEA1* dentro do *HOST A*. Num determinado momento ele precisa se deslocar do seu ambiente atual para um segundo, o *AEA2* que se encontra no *HOST B*. Como pode ser visto na Figura 4.8, após o deslocamento o agente *AG1* não se encontra mais no *AEA1*, deslocando todo seu código e pilha de execução para o ambiente destino, podendo assim prosseguir seu trabalho.

Figura 4.7: Agente situado em um AEA.

Um agente possui uma estrutura interna que representa um itinerário a ser percorrido. Dessa forma, antes de executar uma nova tarefa, o agente preenche seu itinerário que deve

Figura 4.8: Agente transferindo-se de um AEA a outro.

ser público a todos os outros objetos do Minador. Isto permite, por exemplo, que antes que um AEA seja colocado num estado de inatividade, o mesmo consiga avisar a todos os agentes que pretendiam viajar até ele para procurar uma rota alternativa. Para executar determinadas tarefas, um agente terá de cumprir com um itinerário fixo, como está mostrado na Tabela 4.2.

Itinerário	Motivo
AEA 1	TAREFA A
AEA 4	TAREFA A
AEA 6	TAREFA A
AEA 8	DORMIR
...	...

Tabela 4.2: Exemplo de um itinerário a ser seguido por um agente.

Deve existir uma política bem definida de segurança com a qual todos os objetos do sistema (e.g. agentes, AEA) devam concordar e usar, pois agentes “maliciosos” podem tentar entrar no sistema se passando por outros agentes. Assim, a partir da política de segurança, os recursos que um agente pode acessar serão limitados, bem como suas ações. Isso garante que todo e qualquer agente que penetre no sistema não cometa nenhuma ação de vandalismo, como apagar arquivos importantes, iniciar execuções indesejáveis, ou seja, ações que comprometam a operação do computador hospedeiro.

4.7.1.4 Persistência

A característica de persistência envolve três conceitos diferentes no Minador: história, base de relacionamentos e persistência de sistema. Os agentes devem possuir implementado estas três características básicas para operar adequadamente dentro do sistema.

Um agente possui uma estrutura que representa sua história de “vida”. O que precisa ser registrado deve ser muito restrito. Apenas as informações sobre datas, locais e descrição das atividades (tarefas) pertinentes que foram desenvolvidas e o histórico de viagens feitas precisam estar armazenadas. Tais informações serão necessárias para que, por exemplo, um agente consiga colaborar com outros que estiverem executando atividades similares ou sugerir aos usuários o que fazer quando uma mesma operação for requisitada. Nas Tabelas 4.3 e 4.4 está representado como seria o histórico de um agente.

Atividade	Local	Data Início	Data Final	Usuário	Viagem
Coletar Dados	AEA 1	10/06/00 15:03:00	10/06/00 15:20:00	Alfredo	V1
Recuperar Dados	AEA 4	10/06/00 15:10:00	10/06/00 15:15:00	SELF	V2
...	...	...	...	...	...

Tabela 4.3: Histórico de vida de um agente

Viagem	Origem	Destino	Data	Motivo
V1	AEA 1	AEA 4	10/06/00 15:10:00	(t)Recuperar Dados
V2	AEA 4	AEA 1	10/06/00 15:15:00	(a)Voltar p/ Origem
...	...	...	...	...

Tabela 4.4: Viagens feitas por um agente

Como pode ser observado nas duas tabelas, o agente *X* executou a tarefa de *Coletar Dados* no dia *10/06/00* dentro do ambiente *AEA1*, o serviço foi requisitado pelo usuário *Alfredo*. Para completar tal tarefa, o agente fez a viagem *V1* para o *AEA4* para executar a tarefa de *Recuperar Dados*, que foi disparada pelo próprio agente. No *AEA4* ele executou e voltou para a origem (*AEA1*) para continuar o processamento. A ação *Voltar p/ Origem* não deve ser cadastrada uma vez que somente as tarefas relacionadas com as atribuições do agente precisam ser armazenadas.

Uma base de relacionamento mantém armazenado todos os objetos do sistema que interagiram diretamente com um agente. Estes objetos podem ser outros agentes, usuários, base de dados e outros aplicativos. A partir dos relacionamentos armazenados, os agentes

podem decidir que ação tomar, ou sugerir a outros agentes tarefas ou ações para serem realizadas. Um esboço de uma base de relacionamento está apresentada na Tabela 4.5.

Objeto	Motivo	Atividades	Data	Canal
(u)Alfredo	Coletar Dados	(ponteiro)	10/06/00 15:03:00	Módulo de Coleta de Dados
(a)Agente Y	Recuperar Dados	(ponteiro)	10/06/00 15:03:00	Direto
...	...	...	...	...

Tabela 4.5: Base de Relacionamentos de um agente

Observando na tabela anterior, o agente *X* possui armazenado todos os objetos com os quais ele interagiu direta ou indiretamente. De modo indireto, ele interagiu com o usuário *Alfredo* por intermédio do *Módulo de Coleta de Dados*. Diretamente, ele interagiu com o *Agente Y*.

Por fim, os agentes devem conseguir persistir no contexto do sistema, chamado aqui por persistência de sistema. Este tipo de propriedade deve garantir que um agente consiga re-ativar após um estado de inatividade, execução interrompida, falha de sistema ou queda de energia.

A persistência de sistema é conseguida através da construção de agentes utilizando-se mecanismos de persistência da linguagem Java. Basicamente, para todos os agentes deve ser gerado um “arquivo” que armazena a classe e as propriedades no momento da execução de um agente. Este arquivo pode ser armazenado no sistema de arquivos do computador ou em uma base de dados.

4.7.1.5 Máquina de Estados Finitos

O que governa as ações de um agente dentro do sistema Minador é sua Máquina de Estados Finitos (MEF). Um agente executa suas atividades sempre em um dos estados existentes na sua MEF (Figura 4.9). Todos os agentes devem implementar a mesma máquina. Isto é possível pois a única diferença entre os tipos distintos de agentes é o que está ocorrendo no estado *Trabalhando*, ou seja, dois agentes podem estar neste mesmo estado porém executam tarefas diferentes, por exemplo, recuperando dados de alguma fonte, processando os dados, entre outros.

Como pode ser visto na figura, o estado inicial de um agente é *Criado*. Uma vez criado, um agente passa para o estado *Acordado* e a partir de então pode passar para os estados *Trabalhando*, *Dormindo* e *Viajando*. O estado final de um agente é o estado *Morto*.

Figura 4.9: Máquina de Estados Finitos dos agentes do sistema.

A estrutura que representa a MEF de um agente é definida e declarada na classe *Agente Base*.

4.7.1.6 Plano de Trabalho

Todo agente deve possuir um plano de trabalho que permita ser preenchido com as tarefas a serem cumpridas. Este plano deve ser flexível o suficiente para garantir que trabalhos sejam inseridos e/ou sejam removidos pelo agente, a partir de mensagens enviadas pelo sistema.

A estrutura de um Plano de Trabalho pode ser implementada sob a forma de uma fila. À medida que os trabalhos forem sendo gerados, eles entrarão numa fila e serão consumidos por ordem de chegada, ou poderão ser executados concorrentemente quando for o caso. Na Tabela 4.6 está ilustrado um Plano de Trabalho de um agente.

Identificação	Dependência	Trabalho	Parâmetros	Ações
1	-	Conectar Base Dados	Servidor, BD, User, Senha	(ponteiro)
2	1	Selecionar Atributos	Tabela X, campo A, campo B	(ponteiro)
3	2	Salvar Dados	Tabela Y	(ponteiro)
...	...	...	...	...

Tabela 4.6: Plano de Trabalho de um agente

Como pode ser observado na Tabela 4.6, um agente deve retirar da fila um trabalho por vez, a não ser que seja possível realizar trabalhos concorrentes. No exemplo, a primeira tarefa é *Conectar Base Dados*. Para completar esta tarefa, um agente deve fazer um conjunto

de ações. Essas ações podem ser encontradas a partir do ponteiro para ações, que no caso são específicas para *Conectar Base de Dados*. Por fim, o campo *Dependência* indica a dependência ou pré-requisitos para se iniciar um trabalho. Como pode ser visto o trabalho *Selecionar Atributos* é dependente do trabalho anterior. Esse campo é útil para checagem de pré-requisitos no caso de execução de tarefas concorrentes. Por exemplo, a atividade de número 2 é dependente da atividade 1 e dessa forma, as duas não podem ocorrer de forma concorrente.

4.7.1.7 Plano de Ação

O Plano de Ação é o roteiro ou *script* de ações a serem seguidas por um agente, a fim de executar um determinado trabalho. A estrutura que representa o Plano de Ação também é uma fila. Na Tabela 4.7 é ilustrado um exemplo de um plano de ação para se realizar a conexão com uma base de dados.

Identificação	Dependência	Opcional	Ação	Parâmetros	Status
1	-	0	Localizar Recurso	Servidor	Completa
2	1	1	Viajar até Servidor	-	Iniciada
3	2	0	Conectar	Base de Dados	Não Iniciada
4	3	0	Efetuar Login	User, Senha	Não Iniciada
5	4	0	Finalizar Trabalho	-	Não Iniciada

Tabela 4.7: Plano de Ação de um agente

Como pode ser visto (Tabela 4.7), cada ação possui uma identificação única, uma dependência e um campo opcional que indica a sua obrigatoriedade. Pela tabela, como o agente pode se encontrar no mesmo ambiente do servidor de base de dados, a ação *Viajar até Servidor* pode ou não ser executada. Assim, o campo *Opcional* está marcado como verdadeiro (1). Os parâmetros a serem usados ou devolvidos também são incluídos. O campo *Status* de cada ação é preenchido à medida que as ações são executadas, podendo conter os valores *Não Iniciada*, *Iniciada*, *Completa* ou *Fracassada*.

4.7.1.8 Canais de Comunicação

A comunicação no universo dos agentes do sistema é feita utilizando-se um Canal de Comunicação de onde os mesmos conseguem enviar e receber mensagens a outros objetos existentes no sistema. O protocolo usado para a comunicação é o protocolo TCP/IP e o mecanismo

de mensagem disponíveis na API Jumping Beans para realizar a comunicação com outros objetos do sistema. Outros protocolos podem ser suportados pelos agentes, ficando como sugestão para trabalhos futuros a análise de outros protocolos. A Figura 4.10 apresenta os componentes básicos do canal de comunicação de um agente.

Figura 4.10: Canais de Comunicação de um agente.

Como pode ser visto, existem três componentes básicos do canal de comunicação: *Receptor*, *Transmissor* e *Buffer*. O *Receptor* serve para receber mensagens vindas do sistema e de outros agentes. O *Transmissor* envia mensagens diretas e indiretas para os agentes, sistema e usuários do sistema. Devido ao fato de poder existir várias mensagens sendo enviadas e recebidas ao mesmo tempo, os agentes possuem uma estrutura de armazenamento temporário, o *Buffer*, possibilitando aos mesmos gerenciar mais mensagens do que eles possam responder.

4.7.1.9 Memória de Trabalho

Para auxiliar um agente na execução de tarefas, ele possui uma estrutura capaz de armazenar, por exemplo, referências a: algumas das atividades já completadas, agentes com os quais se comunicou, usuários para os quais executou alguma tarefa, data da execução das tarefas e os resultados obtidos.

Para armazenar essas informações, é utilizada uma estrutura denominada Memória de Trabalho. Porém, não é possível armazenar grande quantidade de informações na sua memória, pois se fosse permitido, isso poderia tornar o agente um objeto muito grande (em bytes) dentro do sistema, o que inviabilizaria as viagens para outros ambientes, ou até mesmo

a instanciação do agente no sistema.

A solução projetada é garantir espaço mínimo suficiente para cada tipo de informação a ser armazenada. Logo, ao atingir um limite máximo pré-estabelecido, o sistema deve avisar ao agente para salvar seu histórico na base de informações do Minador, que pode ser representada por um arquivo ou uma base de dados. Dessa forma, quando precisar, os agentes podem consultar o seu histórico através deste objeto externo.

4.7.1.10 Identificação Única

Para que o sistema possa enviar mensagens aos agentes, bem como para que os mesmos possam se identificar a cada AEA e outros agentes, estes devem possuir uma identificação única.

No Minador, o número que identifica um agente, é definido como AAAAMMDDNNNN, onde DDMMAA representa o dia (DD), mês (MM) e ano (AAAA) de criação e os quatro últimos dígitos representam um número de série variando de 0001 a 9999. Por exemplo, o número 200009270001 representa o primeiro agente criado no dia 27/09/2000.

4.7.1.11 Política de Segurança

Uma política de segurança deve ser aceita por todos os agentes do sistema, esta política visa garantir que nenhum agente possa causar algum dano aos ambientes ou aos computadores hospedeiros. A política de segurança é representada por uma interface que deve ser implementada por todos os agentes do sistema e pelos ambientes.

Toda vez que um agente tentar acessar algum recurso do sistema ou de uma ambiente este deve seguir algumas restrições. Um exemplo de uma política de segurança que um agente específico deve respeitar está apresentada na Tabela 4.8.

Objetos	Ler	Escrever	Executar	...
<i>Permissões</i>				
Sistema de Arquivos	Sim	Não	-	...
Rede	-	-	-	...
Windows	Não	Não	Não	...
....				...

Tabela 4.8: Política de Segurança

Como pode ser visto, o agente que possuir instanciado a política representada na

Tabela 4.8, conseguirá ler do sistema de arquivos porém não conseguirá escrever no mesmo.

4.7.2 Agente de Dados

Os Agentes de Dados herdam as características e métodos existentes num Agente Base, implementando adicionalmente, outras propriedades que são utilizadas em tarefas específicas, como: Gerenciador de Dados, Gerenciador de *Drivers* JDBC, Gerador de SQL e o Gerenciador de Segurança de Dados. Estes agentes conectam-se a servidores de bases de dados cadastrados pelos usuários no Módulo de Processamento de Dados (Seção 4.6.3).

O Gerenciador de Dados controla todas as ações pertinentes a coleta, processamento e armazenamento de dados. As informações sobre contas (usuário e senha) em diversos servidores de dados (e.g. SGBD) e os *scripts* de comunicação são administradas por este componente. Ainda, todos os agentes devem armazenar os resultados de suas atividades em uma base de dados acessível a outros objetos do sistema.

Para se comunicar com diferentes SGBD (e.g. Oracle, Sybase e Informix) é utilizado um “conector” especial, o *driver* JDBC. Estes conectores são classes Java disponibilizadas pelos fabricantes com o objetivo de permitir que aplicações diversas se conectem a seus sistemas utilizando-se a linguagem Java. Assim, o Gerenciador de *Drivers* JDBC (*Java DataBase Connectivity*) mantém todos os *drivers* que podem ser usados por um agente.

Função	Comando	Sintaxe Básica
Recuperar	Select	SELECT seleção FROM tabela [WHERE condição]
Inserir	Insert	INSERT [INTO] tabela (atributos) VALUES (valores)
Remover	Drop	DROP objeto nome
Deletar	Delete	DELETE [FROM] tabela [WHERE expressão]
Atualizar	Update	UPDATE tabela SET expressão
Criar	Create	CREATE objeto nome
...	...	

Tabela 4.9: Comandos da *Structured Query Language*.

O Gerador de SQL cria comandos a partir das definições da sintaxe da linguagem SQL. Na Tabela 4.9 são apresentados alguns dos comandos dessa linguagem. O Gerador de SQL possui apenas especificações básicas da sintaxe da SQL. Comandos mais especializados podem ser gerados por classes derivadas do Agente de Dados.

O Gerenciador de Segurança de Base de Dados possui, basicamente, a mesma finalidade que o Gerenciador de Segurança de um agente base. Entretanto, foca essencialmente

na segurança de bases de dados utilizadas pelos agentes. Assim, pode-se especificar quais bases de dados, tabelas, registros e campos os agentes podem ler, modificar, remover, entre outros.

A partir de um Agente de Dados, são derivadas subclasses mais especializadas de agentes: Agente Coletor, Agente de Arquivo, Agente de Internet e Agente Processador. Na Figura 4.11 é apresentado o esquema de operações de todos os agentes derivados durante a realização do processo de KDD. O Agente de Dados não participa diretamente do processo, mas serve como uma classe base a todos os outros.

Figura 4.11: Agentes de Dados do sistema Minador.

Na Figura 4.11 pode ser observado que os agentes do tipo Agente Coletor, Agente de Arquivos e Agente de Internet são voltados para bases de dados, arquivos e Web, respectivamente. O Agente de Arquivo e de Internet não fazem a coleta de dados diretamente de SGBDs e, mesmo assim, ambos são derivados do Agente de Dados. Isto se deve ao fato de que todos armazenam as amostras coletadas em bases de dados temporárias. Por isso, eles precisam ter a habilidade de conectar-se a tais repositórios a fim de armazenar os resultados de suas atividades. O mesmo vale para o Agente Processador, pois este tipo de agente recupera as amostras coletadas para processá-las e armazenar os resultados em um repositório de dados transformados.

Nas próximas subseções são detalhados o funcionamento e as propriedades do Agente Coletor, Agente de Arquivos, Agente de Internet e Agente Processador.

4.7.2.1 Agente Coletor

Um Agente Coletor é especializado em fazer a coleta de amostras contidas em repositórios de dados que possam ser consultados através de sentenças SQL (e.g. SGBD). Este tipo de agente possui implementado internamente um Plano de Coleta, Gerador de Consulta, Coletor de Dados e Gerenciador de Coleta.

O Plano de Coleta de um Agente Coletor é composto basicamente por uma estrutura contendo a base de dados, as tabelas, os atributos e os predicados (condições) da consulta. Um exemplo de um plano de coleta está apresentado na Tabela 4.10.

Base de Dados	Atributos	Tabelas	Predicado
Teste	*	Tab1	-
Teste	*	Tab2	-
...	...	...	...

Tabela 4.10: Plano de Coleta de um agente coletor.

O Gerador de Consultas segue a sintaxe do SQL (Tabela 4.11) para construir uma consulta utilizando a sentença SELECT. As consultas são construídas a partir do Plano de Coleta. Por exemplo, a partir do Plano de Coleta ilustrado na Tabela 4.10, o Gerador de Consultas cria a seguinte consulta: *SELECT * from Teste.TAB1*.

SELECT	lista_seleção
[INTO	nova_tabela]
FROM	tabela_fonte
[WHERE	busca_condicional]
[GROUP BY	expressão_group_by]
[HAVING	condição_busca]
[ORDER BY	expressão_order [ASC DESC]]

Tabela 4.11: Cláusulas principais da sentença SELECT do SQL.

O Coletor de Dados chama o Gerador de Consulta para construir consultas SQL. O Gerenciador de Coleta, utiliza também o Plano de Coleta para administrar a execução das consultas, podendo recuperar erros, reexecutar consultas mal sucedidas, entre outros.

4.7.2.2 Agente de Arquivo

Um Agente de Arquivo coleta dados provindo de arquivos do tipo ASCII e binários. Os componentes principais deste tipo de agente de dados são: Plano de Coleta, Parser e Gerenciador de Coleta.

O Plano de Coleta deste agente também está representado por uma estrutura que lista todas as características de uma tarefa de coleta a ser feita. Neste plano pode estar especificado a árvore de diretório, nomes dos arquivos, extensões aceitáveis, atributos a serem coletados, tipos dos atributos, separadores conhecidos, entre outros. Um exemplo desta estrutura é apreseto na Tabela 4.12.

Diretório	Arquivos	Extensões	Atributos	Separador	Tipo de Dados
dirA	(arq1,arq2)	txt	(a1,a2,a3,a4)	:	(int,int,string)
dirB	(arq3)	txt	automático	automático	automático
...	...	...	...	...	...

Tabela 4.12: Plano de Coleta de um agente de arquivos

O *Parser* utiliza as informações contidas no Plano de Coleta para processar os arquivos textos, retirando os símbolos (*tokens*) e gerando um arquivo de resultado para cada arquivo coletado na forma atributo-valor, onde os valores estão separados por um caracter padrão. Caso nenhuma informação sobre os atributos, separadores, tipos de dados e nome dos atributos sejam especificados ou conhecidos, então são utilizados valores padrões pré-definidos. Assim, o *Parser* executa a varredura no arquivo levando em consideração valores padrões e tenta encontrar automaticamente os símbolos nos arquivos.

O Gerenciador de Coleta usa as informações do Plano de Coleta para ajustar outros componentes herdados do agente base como o Plano de Trabalho e o Plano de Ações. Em seguida ele pode dar inicio à etapa de coleta, pegando os resultados das coletas e exportando os arquivos resultantes para dentro de uma base de dados, utilizando sentenças SQL específicas de inserção ou de atualização de dados.

4.7.2.3 Agente de Internet

Para recuperar dados de páginas de *Web*, um tipo especial de agente de dados foi especificado, o Agente de Internet. Os componentes destes agentes são: Plano de Coleta, Conector de Internet, Parser de HTML e um Gerenciador de Coleta.

O Plano de Coleta armazena todos as informações necessárias para realizar a busca por informações contidas em documentos HTML, como a URL (*Uniform Resource Locator*) do site, porta, páginas, objetos a serem coletados (e.g. apontamentos), configuração do parser, entre outros. Estes dados podem ser configurados através do Gerenciador de Descobrimento via Interface de *Web*.

As páginas podem ser coletadas na íntegra e armazenadas num repositório. Outra possibilidade é utilizar o Parser HTML para limpar as *tags* desnecessárias, ou mesmo, recuperar dados na forma de atributo-valor. Porém, isto depende especificamente da configuração do parser, pois não existe uma configuração única que consiga recuperar informação de páginas diferentes. Uma opção mais viável para se conseguir este tipo de funcionalidade é a utilização de outro padrão, conhecido por XML (*eXtensible Mark-up Language*) que possibilita a organização dos dados nas páginas, permitindo que os dados sejam recuperados como se estivessem organizados em uma “base de dados”.

O Gerenciador de Coleta usa as informações do Plano de Coleta para ajustar outros componentes herdados do agente Base como o Plano de Trabalho e o Plano de Ações. Assim, este gerenciador pode controlar o trabalho de coleta de um agente, armazenar resultados, entre outros.

4.7.2.4 Agente Processador

Um Agente Processador possui habilidades que permitem ao mesmo executar diversas tarefas sobre os dados, como:

- *transformações*: os atributos podem ser convertidos em outros tipos e os valores dos atributos podem ser modificados;
- *limpeza*: pode ser necessário eliminar atributos com uma porcentagem alta de valores nulos ou inválidos;
- *redução*: algumas vezes é necessário reduzir o número de atributos ou o número de registros de uma tabela, para que possam ser processados por uma ferramenta de mineração, por exemplo;
- *adição*: novas colunas podem ser criadas a partir da combinação das existentes;

- *padronização*: valores de um atributo devem estar sempre no mesmo padrão, assim, valores com o mesmo significado mas com valores múltiplos devem ser padronizados;
- *agregações*: funções de agregação podem ser aplicadas, como a média, desvio padrão, mediana, máximos, mínimos, somatório e contador.

Os componentes existentes nesta classe de agente são um Plano de Processamento, um Processador de Dados e um Gerenciador de Transformação. A fonte de dados destes agentes é a base de amostras coletadas que é constituída dos resultados obtidos pelos agentes coletores, de arquivos e de Internet.

O Plano de Processamento é uma estrutura contendo informações sobre os processamentos a serem realizados. Logo, dados sobre o tipo de processamento (e.g. remoção), as tabelas, os atributos, os valores e os parâmetros de transformação ficam guardados neste plano. Essa estrutura deve armazenar as configurações específicas a cada tipo de processamento.

Ao fazer uma varredura pelo Plano de Coleta, o Processador de Dados extrai as informações necessárias para criar sentenças SQL usadas para processar os dados da base de amostras coletadas. Os resultados gerados são armazenados nesta mesma base em tabelas diferentes. Os agentes não devem modificar as amostras, a não ser que seja especificado pelo usuário.

Finalmente, o Gerenciador de Transformação utiliza as informações do Plano de Processamento para criar uma seqüência de processamento e configurar o Plano de Trabalho e Plano de Ação herdados dos agentes de Dados e Base.

4.7.3 Agente Minerador

Uma vez tendo sido processados, os dados devem ser formatados de acordo com as necessidades das ferramentas de mineração. O Agente Minerador é responsável por criar os arquivos utilizados por estas ferramentas, como pode ser visto na Figura 4.12. Os agentes somente são responsáveis por criar os arquivos necessários e não pela execução das minerações diretamente. Para a execução da mineração de dados pode ser utilizado uma ou um conjunto de ferramentas externas de mineração. Estas ferramentas por sua vez geram modelos e padrões dos dados.

Figura 4.12: Agente Minerador do sistema Minador.

A maioria das ferramentas e algoritmos de mineração conseguem ler apenas arquivos no formato ASCII. Normalmente, dois tipos básicos de arquivos são utilizados: configuração e dados. O primeiro arquivo apresenta especificações sobre a estrutura e tipos dos dados, os separadores de dados, comandos a serem executados, configurações de mineração, entre outros. O arquivo de dados possui os dados que podem estar num formato específico, possuir separadores, entre outros. Vale ressaltar que dependendo da ferramenta o arquivo de dados pode ser conhecido por outro nome, porém, a sua funcionalidade permanece a mesma. Um exemplo de arquivo de configuração e de dados está apresentado na Tabela 4.13 e Tabela 4.14, respectivamente.

input	{
file	"funcionarios.data";
string	funcionario;
string	cargo;
float	salario;
	}

Tabela 4.13: Exemplo de um arquivo de configuração utilizado pela ferramenta MineSet.

Joao	Gerente de Vendas	1000.00
Maria	Diretora de Marketing	5000.00
Thais	Designer	3500.00
Helena	Secretária	800.00
...	...	...

Tabela 4.14: Exemplo de um arquivo de dados utilizado pela ferramenta MineSet.

No arquivo de configuração apresentado na Tabela 4.13 estão indicados o nome do

arquivo de dados, *funcionarios.data* e os atributos *funcionario*, *cargo* e *salario* dos tipos de dados *string*, *string* e *float*, respectivamente. O conteúdo do arquivo de dados que é referenciado no arquivo de configuração é mostrado na Tabela 4.14.

Os componentes que um Agente Minerador implementa a partir do Agente Base são: Gerenciador de Algoritmos, Plano de Mineração, Gerador de Arquivos e Executor de Lote. O projeto destes agentes permite que outros componentes sejam acrescentado, por exemplo, um possível Minerador de Dados pode ser adicionado a sua classe, permitindo, num momento futuro, que os agentes também possam executar minerações.

O Gerenciador de Algoritmo administra para quais algoritmos ou ferramentas é possível criar arquivos. Para tanto, o gerenciador deve possuir registrado internamente em sua estrutura os nomes dos algoritmos, a estrutura, a nomenclatura e extensões dos arquivos necessários, entre outros. Assim, para se cadastrar um novo algoritmo, sua estrutura deve ser incrementada com as características deste algoritmo.

O Plano de Mineração é uma estrutura onde podem ser inseridas informações sobre as tarefas de mineração, os algoritmos/ferramentas para os quais serão criados arquivos, as fontes de dados, os atributos, os valores, a parametrização dos algoritmos, taxas de erros, entre outros.

O Criador de Arquivos utiliza as informações no Plano de Mineração para iniciar o processo de criação dos arquivos necessários. Os arquivos de configuração são criados somente se o algoritmo estiver catalogado, no Gerenciador de Algoritmo.

Alguns algoritmos e ferramentas podem ser executados a partir de arquivos de lote. Dessa forma, é possível configurar um agente para executar um arquivo de lote disponível para iniciar o processamento de mineração de uma ferramenta. O componente que realiza este processamento em lote é o Executor de Lote.

Os modelos e padrões encontrados do processo de *Data Mining* podem estar representados em arquivos ASCII, figuras, animações, gráficos, relatórios, entre outros. Logo, os usuários devem configurar as ferramentas para salvar os resultados em um diretório padrão de onde possam ser acessados pelo sistema Minador.

4.7.4 Agente Distribuidor

O Agente Distribuidor faz a distribuição dos resultados obtidos pelas ferramentas de mineração utilizadas. Na Figura 4.13 está apresentada uma visão geral do esquema de operação dos agentes distribuidores. Estes resultados são representados por um componente chamado de Objeto de Conhecimento. Os usuários podem receber os resultados via correio eletrônico ou acessar o repositório de informações do sistema, por exemplo. Os componentes pertencentes ao Agente Distribuidor são: Objeto de Conhecimento, Plano de Distribuição, Mapa de Distribuição e Distribuidor.

Figura 4.13: Agente Distribuidor do sistema Minador.

Os Objetos de Conhecimentos são estruturas que armazenam algumas informações sobre os resultados gerados pelas ferramentas de mineração, como: local do objeto, tamanho em bytes, nome do objeto (e.g. nome do arquivo), extensão e tipo do resultado (e.g. gráfico). Estes objetos apontam para os resultados contidos em um diretório ou em uma base de dados.

O Plano de Distribuição contém informações que especificam quais usuários devem receber os Objetos de Conhecimento resultantes, data e horário de entrega, pareceres de usuários, quais objetos serão enviados, entre outros.

O Mapa de Distribuição é construído a partir do Plano de Distribuição. Nele estão contidos endereços e nomes para onde serão distribuídos os Objetos de Conhecimento.

O Distribuidor ou Máquina de Distribuição utiliza o Plano e o Mapa de Distribuição para iniciar os componentes herdados do Agente Base que coordenam a execução de tarefas de um agente. Em seguida, os distribuidores podem executar a distribuição dos objetos das seguintes formas: deixando referências dos mesmos no Gerenciador de Descobrimento, enviando-os via e-mail para os usuários ou enviando mensagens de aviso aos usuários sobre a disponibilidade dos resultados.

4.8 Ambiente de Execução de Agentes

Os Ambientes de Execução de Agentes são os locais onde os agentes executam suas atividades e ficam inativos (“dormindo”). Os AEAs também funcionam como estações de transporte e de comunicação dos objetos do sistema, que podem ser agentes, mensagens e arquivos.

Os agentes executam suas tarefas dentro dos AEAs. Isto garante que somente agentes do sistema estarão utilizando os recursos existentes. Ademais, como os computadores e sistemas operacionais restringem o acesso a seus recursos, um agente não consegue entrar livremente nos mesmos. Assim, um ambiente deve estar rodando na mesma máquina onde um agente pretenda ir, pois este AEA reconhece o agente como sendo do Minador e permitir a sua execução e utilização dos recursos existentes no computador.

Os agentes podem trafegar livremente por entre os ambientes, desde que possuam autorização e sigam a política de segurança estabelecida pelo sistema Minador. Dessa forma, um agente pode transportar-se de um AEA a outro visando utilizar um determinado recurso na máquina (e.g. base de dados), assim que completar sua tarefa, o agente pode, por exemplo, deslocar-se para outra máquina ou ir “dormir” por um período finito de tempo.

Uma representação da arquitetura dos AEAs e seus componentes principais pode ser observada na Figura 4.14. Nela, é possível verificar que cada ambiente possui uma estação e um canal de comunicação. As estações possuem um mecanismo de envio e recebimento de objetos para outros ambientes. Os canais de comunicação são elementos físicos que auxiliam o tráfego dos objetos do sistema e captam eventos ocorridos em outros ambientes. Para manter uma boa qualidade de na comunicação do canal, o mesmo possui um *buffer* onde armazena as mensagens que chegam e ainda não foram processadas.

Figura 4.14: Ambientes de Execução de Agentes.

O Servidor de Ambiente, ou Servidor de Agência, gerencia todas as trocas de mensagens entre os AEAs, mantém os nomes e endereços IP de todos os AEAs, agentes e objetos que estejam executando em uma AEA. Através deste servidor é possível encerrar sumariamente a execução de objetos, revogar AEAs, gerenciar permissões, limites de acesso dentro dos ambientes. Em suma, o Servidor de Agência é utilizado para facilitar o gerenciamento e controle dos AEAs e dos agentes.

Os AEAs são construídos através da API Jumping Beans, fornecida pela empresa Ad Astra Engineering. Esta interface de programação possui facilidades para a construção de aplicações móveis genéricas, garantindo uma boa flexibilidade e facilidade na construção de aplicações para este fim. Ainda, como a API utiliza a linguagem Java, é possível executar os ambientes em qualquer plataforma e sistema operacional. Na Figura 4.14 estão apresentados alguns dos sistemas onde os AEAs conseguem executar.

4.9 Consideração Sobre Implementações e Testes Realizados

Das idéias apontadas neste trabalho, algumas foram implementadas para realização de testes e experimentos (Rezende et al., 2000). O objetivo destes experimentos foi verificar a viabi-

lidade de algumas propriedades proposta no sistema Minador. Assim foram implementados classes de AEA, mobilidade de objetos entre os mesmos, coleta automática de dados e tratamento de dados. Com isto, também pôde ser feito uma análise de viabilidade dos softwares usados na possível implementação de um sistema completo.

4.9.1 Peculiaridades da Implementação

O desenvolvimento de uma aplicação leva em considerações diversos fatores, por exemplo, plataformas e versões suportadas. Para implementar algumas das funcionalidades do Minador existem alguns fatores específicos de implementação e execução a saber:

- *Plataformas de desenvolvimento e execução*: as plataformas usadas para o desenvolvimento dos experimentos foram Windows NT e Solaris 7. Enquanto que as plataformas usadas para execução dos mesmos foram Windows NT/2000, Solaris 7, Irix 6.5 e Linux.
- *Sistema Gerenciador de Base de Dados*: o servidor de base de dados usado para armazenar os dados operacionais é o Oracle 8i.
- *Versão do Java Development Kit (JDK)*: a implementação foi feita utilizando a versão 1.2.2 do Java.
- *Ferramenta IDE*: o ambiente de desenvolvimento utilizado foi JBuilder¹⁰ Professional (versões 3.0 e 3.5) da empresa Inprise Inc.
- *API do AEA*: a versão da API JumpingBeans usada foi a 1.2 com possibilidade de se executar oito ambientes e tendo um servidor.
- *Servidor de Web*: o servidor de Web utilizado é o Apache ¹¹ 1.3.6 para plataforma Windows NT.
- *Browsers*: os *browsers* usados para testar a interface gráfica são o Internet Explorer 5 e Netscape 4.7.
- *Servlet*: a interface de Web foi construída usando-se o *engine* Tomcat¹² que é compatível com Servlet API 2.2 e JSP 1.1.

¹⁰<http://www.borland.com/jbuilder>

¹¹<http://www.apache.org.br>

¹²<http://jakarta.apache.org>

- *Drivers JDBC*: como o SGBD utilizado é o Oracle8i, é usado o *driver* JDBC (versão 1.1.1) pertencente a este SGBD.
- *Suites para Mineração de Dados*: os arquivos de configuração e de dados são gerados para o *suite* MineSet com versão 2.6 ou acima.

4.9.2 AEA

Um componente básico para o Minador são seus Ambientes de Execução de Agentes, pois estes representam o mundo onde os agentes executam os trabalhos, os eventos são disparados e permitem que novos computadores sejam inseridos no sistema. Assim, achou-se interessante verificar a viabilidade da construção desses ambientes.

Algumas classes de AEA e um servidor de AEA foram criadas. Este último era o responsável por gerenciar todos os outros ambientes existentes, bem como inserir novos ambientes de forma *on-line*. A implementação dos ambientes não possui nenhuma interface ao usuário, ela é considerada como sendo parte da infra-estrutura para o sistema. Os ambientes criados executaram com sucesso em diferentes plataformas como WindowsNT, Windows95/98, Solaris, Irix e Linux. Foi possível inserir e remover ambientes e objetos. Armazenar o estado de cada ambiente criado, de forma que mesmo se houvesse uma pane no sistema operacional o estado anterior ao problema do ambiente foi mantido.

4.9.3 Coleta Automática de Dados e Tratamento dos dados

Um experimento foi feito visando coletar automaticamente dados da Internet (Rezende et al., 2000). Os objetos criados eram capazes de buscar por determinadas páginas e recuperar dados especificados pelos usuários através de uma pequena interface de consulta. Estes dados eram trazidos e armazenados numa base de dados Oracle.

Uma vez tendo sido coletados os objetos permitiam que o usuário pudesse transformar os dados através de uma interface. A criação automática de arquivos de dados e de configuração suportados pelo *suite* MineSet também foi verificada. Estes testes permitiram identificar novas possibilidades ao sistema como a mineração de textos da *Web*.

4.10 Protótipo Elaborado

Um protótipo foi criado a fim de se ter uma visão mais elaborada do possível funcionamento do sistema Minador. Algumas das características implementadas permitem que possam ser testadas a execução dos agentes, conexão com fontes de dados, seleção e coleta de dados e a criação de arquivos usados para a mineração.

Nas Tabelas 4.15, 4.16, 4.17 e 4.18 estão apresentadas as características implementadas neste protótipo para a Interface de *Web*, Gerenciador de Descobrimento, Agentes e AEA, respectivamente. Também são apresentadas as características restantes para se construir o sistema completo.

Na Tabela 4.15 estão apresentados os componentes implementados na Interface de *Web*. Os usuários utilizam páginas HTML para acessar as funcionalidades existentes no protótipo. Estas páginas possuem *Forms* que são formulários com campos para serem preenchidos e, uma vez feito o preenchimento, as informações contidas são processadas pelo Servidor de Aplicação, um programa Servlet, que é executado no mesmo computador onde fica o servidor de *Web*. Este programa possui a lógica do funcionamento da interface gráfica do usuário do Minador.

Interface de Web		
Componente	Protótipo	Completo
GUI	HTML	JSP Applet
Servidor de Aplicações	Servlet	JSP/Servlet CGI e-mail

Tabela 4.15: Implementação do Protótipo de Interface de *Web*

Na Tabela 4.16 estão apresentadas as propriedades implementadas para o Gerenciador de Descobrimento. O usuário acessa os módulos existentes através de páginas HTML. No módulo de Configuração de Instâncias, pode-se criar novas instâncias, descrever os objetivos do processo e criar uma lista das atividades (etapas) que serão executadas. Foram implementados os componentes necessários para configurar fontes de dados, informações para conexão com um servidor de base de dados, selecionar os atributos, editar sentenças SQL, definir a porcentagem de dados da amostra, agendar coletas e visualizar os dados.

Para criar os arquivos de configuração e de dados, foram implementados os compo-

nentes para selecionar os tipos de dados existentes no MineSet, selecionar os atributos e opções para criar arquivos automaticamente.

Gerenciador de Descobrimento		
Componente	Protótipo	Completo
Configuração de Instância	Criação de Instâncias Nome da Instância Definição de Objetivos Editor de Atividades	Papéis dos Usuários Configuração de AEA Criação de Contas de Usuário
Monitor do Sistema	Status do Processo Execução de Instância Log	Logs Gerenciador de Recursos Editor de Parecer Editor de Mensagens
Processamento de Dados	Conexão de Base de Dados Editor de Coleta Visualização de Dados Configuração de Fontes de Dados Seleção de Atributos Editor de SQL Amostragem Agenda de Atividades	Seleção/Criação de Base Temporária Verificador de Sintaxe SQL Sumarização Editor de Filtros Editor de Agregações Editor de Amostragem Parser Segurança
Mineração	Seleção de Tipos Seleção de Atributos Criação Automática de Arquivos	Seleção de Algoritmo Seleção de Ferramentas Parametrização Editor de Arquivos de Configuração Editor de Arquivos de Dados Editor de Arquivos de Lote <i>Holdout</i> e Resampling
Distribuição	Visualizador de Instâncias Histórico	Permissão de Usuários Visualizador de Resultados Editor de Distribuição Objetos de Conhecimento

Tabela 4.16: Implementação do Protótipo do Gerenciador de Descobrimento.

Os componentes necessários para testar as operações dos agentes foram implementados, com pode ser visto na Tabela 4.17. Desse modo, o Agente Base possui as propriedades de reatividade, mobilidade, plano de trabalho, canal de comunicação, máquina de estados finitos, identificação única e plano de ação. Um Agente de Dados possui implementado o Gerenciador de Dados, Gerador de SQL e Gerenciador de JDBC. Para o Agente Coletor foi implementado um Plano de Coleta, Gerador de Consulta e Coletor de Dados. Finalmente, o Agente Minerador possui um Plano de Mineração e um Gerador de Arquivos.

Como os AEAs foram criados utilizando-se a API JumpingBeans, a maioria dos com-

Agentes		
Tipo	Protótipo	Completo
Agente Base	Reatividade Mobilidade Plano de Trabalho Canais de Comunicação Máquina de Estados Finitos Identificação Única Plano de Ação/Reação	Pró-Atividade Persistência Gerenciador de Segurança Memória de Trabalho Política de Segurança
Agente de Dados	Gerenciador de Dados Gerador de SQL Gerenciador de JDBC	Gerenciador de Segurança de Dados
Agente Coletor	Plano de Coleta Gerador de Consulta Coletor de Dados	
Agente de Arquivo		Plano de Coleta Parser de Arquivos Gerenciador de Coleta
Agente de Internet		Plano de Coleta Internet Parser de HTML Gerenciador de Coleta
Agente Processador		Plano de Processamento Processador de Dados Gerenciador de Transformação
Agente Minerador	Plano de Mineração Gerador de Arquivos	Gerenciador de Algoritmos Processador de Lote
Agente Distribuidor		Objetos de Conhecimento Plano de Distribuição Mapa de Distribuição Distribuidor

Tabela 4.17: Implementação dos Agentes do Protótipo.

Ambiente de Execução de Agente	
Protótipo	Completo
Estação	Política de Segurança
Canal de Comunicação	Gerenciamento Remoto

Tabela 4.18: Implementação dos AEA.

ponentes já se encontrava implementada. Para adaptar os AEAs ao sistema Minador foram criados a estação de recebimento de agentes e o canal de comunicação. Estes dois componentes permitem gerenciar os agentes que entram dentro de cada ambiente.

O protótipo possui uma base de dados operacional que serve para armazenar e processar todas as operações em andamento ou realizadas. As informações referentes ao proces-

samento de cada instância ficam armazenadas neste repositório.

Um conjunto de tabelas é usado para armazenar as descrições do domínio, objetivos, usuários, AEAs, agentes e atividades pertencentes às instâncias criadas. A base de dados, bem como outros módulos, é acessada indiretamente pelos usuários através da Interface de *Web*. O modelo físico (para o SGBD Oracle) deste repositório está apresentado na Figura 4.15.

Figura 4.15: Modelo físico da base de dados do Minador.

Na Figura 4.15 pode-se observar as informações armazenadas a respeito de uma instância. A tabela *DOMINIO* guarda as descrições sobre os objetivos do processo. As tabelas

PROCESSAMENTO e *MINERACAO* mantêm as informações de coleta e criação de arquivos, respectivamente. As informações sobre os agentes (tabela *AGENTE*) e AEA (tabela *AEA*) utilizados também são registradas. Finalmente, as atividades especificadas (*ATIVIDADE*) e os usuários (*USUARIO*) que participam daquela instância ficam mantidos na base de dados.

Os dados que são recuperados pelos agentes durante a coleta de dados podem ser armazenados no sistema em tabelas temporárias, que são usadas enquanto uma instância é processada. No final do processamento, estas tabelas precisam ser removidas para liberar espaço em disco.

O protótipo apresenta as características necessárias para a se conseguir a implementação de uma primeira versão do sistema que seja operável.

4.11 Considerações Finais

Neste capítulo foi apresentado a proposta de um sistema baseado em agentes de software para apoiar o processo de Extração de Conhecimento de Base de Dados.

O Minador apresenta-se com uma arquitetura distribuída distribuída, multiagente, independente de plataforma e cuja interface está baseada em páginas de *Web* capazes de serem acessadas por qualquer computador que possua acesso à Internet e tenha um *Browser* instalado.

Os usuários devem interagir com o sistema através de sua Interface de *Web*. A partir dela é possível configurar cada etapa do processo de KDD, definindo fontes de dados, repositórios de amostras, coletas de dados, transformação nos dados, arquivos de mineração e forma de distribuição de resultados. Não existe comunicação direta entre os agentes e os usuários, esta comunicação existe de forma indireta utilizando-se como meio o Gerenciador de Descobrimento, onde são especificadas os requisitos para a execução de uma instância.

Pode ser notado que o sistema não possui a intenção de operar com processos muito complexos, onde a aprendizagem é o ponto forte, mas sim, ser um sistema que esteja apto a realizar tarefas simples e rotineiras para servir como um assistente ao usuário especialista.

Dois trabalhos nesta mesma linha, execução de KDD utilizando-se agentes de software, foram estudadas: Little-JIL e InfoSleuth. Algumas das idéias contidas nestes projetos foram utilizadas e foram de grande utilidade para a elaboração do Minador. Existem dife-

renças entre as duas abordagens e o sistema proposto. Estas diferenças representam uma nova abordagem para se tentar trabalhar com agentes de software em um ambiente de rede distribuída para auxiliar a extração de conhecimento.

A partir de novos estudos de necessidades, outros agentes podem ser projetadas, ambientes com funcionalidades diferentes, outras ferramentas ou metodologias podem ser usadas em conjunto com o Minador. Ou seja, o sistema foi proposto para que fosse possível ser adicionado novas características, sendo, dessa forma, um sistema bastante flexível, diferentemente dos outros projetos estudados.

As contribuições oferecidas no sistema Minador que o diferencia dos outros dois sistemas citados, são: tipologia de agentes baseadas num agente Base; tratamento e gerenciamento dos objetos e agentes existentes no sistema através dos Ambientes de Execução de Agentes e Gerenciamento feito através de módulos que podem ser adicionados ao sistema. Estas características podem ser melhoradas e detalhadas mais a fundo a fim de auxiliar a criação de sistemas multiagentes para apoiar a execução de processos de KDD.

Capítulo 5

Conclusões e Propostas para Trabalhos Futuros

A grande quantidade de dados, processada e armazenada por sistemas computacionais, requer que novas técnicas e metodologias capazes de auxiliar a análise e o entendimento dos dados sejam produzidas.

O uso de redes de computadores pode aumentar em muito a capacidade de transmissão, processamento e, por conseguinte, o volume de informações que podem estar em diferentes formatos. Além disso, algumas das técnicas que, normalmente, são utilizadas para se extrair conhecimento de repositórios locais ou interligados em uma rede local não são adequadas para analisar as informações contidas na Internet diretamente.

Quanto mais distribuídos estiverem os dados, mais pessoas dependerão deles para tomar decisões em tempo hábil e de forma precisa. Assim, todas elas devem participar do processo de extração de conhecimento.

Muito do trabalho que é feito para se descobrir conhecimento nos dados, principalmente no que se refere à coleta e tratamento de informações, pode ser repetitivo e estafante. Dessa forma, num projeto envolvendo um grande número de pessoas, é essencial que existam formas automáticas para executar as atividades “não intelectuais”. Isto pode liberar os analistas para realizarem tarefas mais importantes ao processo de extração e garantir uma melhor padronização na execução das demais tarefas.

Nesse ínterim, é do interesse de organizações e da comunidade científica encontrar meios que adaptem as tecnologias existentes para serem aplicadas num paradigma de com-

putação em rede. Assim, este trabalho foi focado na busca de uma possível solução às carências existentes na área de KDD.

A partir do estudo de pesquisas feitas na comunidade científica (Capítulos 2 e 3), percebeu-se que o uso de agentes de software pode ser uma alternativa bastante interessante na tentativa de se adaptar algumas técnicas da área de KDD ao ambiente de redes de computadores.

Neste contexto, o objetivo maior deste trabalho de Mestrado foi propor a idéia de um sistema multiagente, denominado Minador (Capítulo 4), usado para auxiliar o gerenciamento e execução do processo de Extração de Conhecimento de Base de Dados.

O Minador possui agentes de software que foram projetados para resolver problemas como a integração de ferramentas usadas para executar as etapas do processo de KDD. Isto permite que tarefas, como a coleta e tratamento de dados, possam ser feitas de modo autônomo pelos agentes. Ainda, os arquivos necessários às ferramentas de mineração são criados. Finalmente, os resultados obtidos em cada etapa podem ser distribuídos para todos os usuários envolvidos (Oliviera and Rezende, 2001).

Para garantir que os envolvidos num projeto de KDD possam utilizar o sistema independente de plataforma e local onde estejam, toda a interface ao usuário foi projetada para ser disponibilizada através da *Web*. A partir desta interface, o usuário pode monitorar as atividades sendo realizadas em cada instâncias e configurar novas instâncias.

A implementação do sistema contendo todas as funcionalidades e propriedades propostas, pode se estender por vários meses e consumir muito recursos humanos. Apesar de não ser o objetivo do trabalho, foram executados alguns testes para verificar a viabilidade de algumas funcionalidades bem como a utilização de algumas ferramentas. Estas pequenas implementações foram usadas para a realização de experimentos como a coleta de dados automática da Internet, verificação da mobilidade, entre outros já citados anteriormente. Com os testes realizados foram conseguidos alguns indícios de que a construção do sistema pode ser viável para auxiliar projetos ou experimentos isolados envolvendo KDD.

Como proposta de trabalhos futuros, visando aprimorar mais o sistema, as seguintes características podem ser inseridas ou modificadas:

- *Ambiente de Execução de Agentes*: novas API podem ser usadas para a criação de ambientes de execução dos agentes, por exemplo, IBM Aglets (Lange and Oshima,

1998). Um ambiente próprio também pode ser criado através da API RMI (*Remote Method Invocation*) do Java. Apesar de ser mais complexa, esta última alternativa é mais interessante, pois pode-se criar um ambiente de execução de agentes especializado para o processo de KDD.

- *Base de Conhecimento*: uma base de conhecimento pode auxiliar os agentes na aprendizagem e interação com outros objetos do sistema Minador. Um agente poderia possuir os casos mais importantes incluídos neste repositório, o que facilitaria a tomada de decisões a partir de casos passados. Outra alternativa interessante, é cada agente possuir uma pequena base de conhecimento, que pode garantir uma maior independência para os agentes.
- *Ontologia*: pode-se criar uma ontologia específica para o processo de KDD, que fosse usada por diversos agentes dentro do sistema. Isto permitirá compartilhar conhecimentos comuns entre os agentes. Nesse sentido, podem ser usadas linguagens para a construção e compartilhamento de ontologias como a Ontolingua (Farquhar et al., 1996).
- *Linguagem de Comunicação*: uma linguagem de comunicação de agentes pode ser usada, permitindo que os agentes do sistema consigam trocar conhecimentos e cooperar uns com os outros, bem como permitir a comunicação com outros sistemas que utilizem a mesma linguagem. Diversas alternativas estão disponíveis como (Wooldridge and Jennings, 1994; Muller et al., 1997): KQML, ARCOL, Telescript, Agent-0, PLACA, LOGOS, entre outras.
- *XML*: a interface de *Web* pode ser projetada para suportar o formato XML. Este formato permite criar marcadores com significados nas páginas da *Web* e, desse modo, os resultados obtidos em cada etapa do processo podem ser mais bem gerenciados e compartilhados com todos os usuários.
- *Mineração de Dados*: os agentes mineradores podem possuir componentes que implementem alguns dos algoritmos usados para fazer *Data Mining* e, dessa forma, os agentes conseguiriam realizar a mineração diretamente nos dados. Podem ser usadas API para implementar os algoritmos de mineração, por exemplo, a ferramenta Weka (Witten and Frank, 1999) possui diversos algoritmos implementados na linguagem Java.

- *Mineração de Textos*: como muitas das informações contidas na Internet estão no formato de textos, é possível adaptar o sistema Minador para realizar o processo de *Text Mining* (Zanasi, 1998) extraindo conhecimentos embutidos em documentos não estruturados. Para isso, deve-se estender a capacidade do sistema para conseguir coletar e tratar textos, sendo necessária a especificação de novos módulos e tipos de agentes para auxiliar na mineração de textos.

Um estudo de caso futuro pode ser feito em conjunto com os pesquisadores do Programa de Melhoramento Genético da Raça Nelore (PMGRN) (Lôbo et al., 1999) que vêm, já há alguns anos, fazendo análises estatísticas dos dados sobre rebanhos bovinos brasileiros para predizer quais animais serão os melhores reprodutores e matrizes.

A base de dados do PMGRN possui, atualmente, dados de mais de 170.000 animais. Estes dados contêm informações sobre o nascimento, pesagens, vacinação, alimentação, entre outros, dos touros e matrizes de 88 fazendas cadastradas no programa. Algumas pesquisas já vêm sendo feitas em parcerias com os pesquisadores (Rezende et al., 1998a) para se obter conhecimento desses dados. Pode-se utilizar o sistema Minador para auxiliar no processo de KDD, verificando-se a utilidade do sistema num domínio de aplicação real.

Bibliografia

- Aha, D., Kibler, D., and Albert, M. (1991). Instance-Based Learning Algorithms. *Machine Learning*, (6):37–66.
- Ambite, J., Ashish, N., Knoblock, C., Minton, S., Modi, P., Muslea, I., Philpot, A., and Tejada, S. (1998). Modeling Web Sources for Information Integration. In *Modeling Web Sources for Information Integration*, pages 211–218. AAAI Press / The MIT Press.
- Ashish, N., Knoblock, C., and Levy, A. (1997). Information Gathering Plans with Sensing Actions. In *Fourth European Conference on Planning 1997*.
- Bayardo, R. (1997). Infosleuth: Agent-based Semantic Integration of Information in Open and Dynamic Environments. Tucson, AZ. ACM SIGMOD Conference on Management of Data.
- Bhandari, I., Colet, E., Parker, J., Pines, Z., Pratap, R., and Ramanujan, K. (1997). Advanced Scout: Data Mining and Knowledge Discovery in NBA Data. *Data Mining and Knowledge Discovery*, 1(1):121–125.
- Bontempo, C. and Zagelow, G. (1998). The IBM Data Warehouse Architecture. *Communications of the ACM*, 41(9):38–48.
- Brachman, R. J. and Anand, T. (1996). *The Process of Knowledge Discovery in Databases: A Human-Centered Approach*, pages 37–58. In (Fayyad et al., 1996a).
- Bradley, P., Fayyad, U., and Mangasarian, O. (1998). Data Mining: Overview and Optimization Opportunities. Technical Report MSR-TR-98-04, Microsoft Research Report, Redmond, WA.

- Bradshaw, J. (1997a). *An Introduction to Software Agents*, pages 3–46. In (Bradshaw, 1997b).
- Bradshaw, J. (1997b). *Software Agents*. AAAI Press / The MIT Press, Cambridge, MA.
- Braga, A. P., Carvalho, A. P. L. F., and Ludermir, T. B. (2000). *Redes Neurais Artificiais: Teoria e Aplicações*. LTC, Belo Horizonte, MG.
- Chaudhuri, S. and Dayal, U. (1997). An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1).
- Davies, W. and Edwards, P. (1995). Agent-Based Knowledge Discovery. In *AAAI Spring Symposium on Information Gathering*, pages 34–37.
- Ebecken, N. F. F., editor (1998). *Data Mining*. WIT Press - Computational Mechanics Publications, England.
- Elder IV, J. and Pregibon, D. (1996). *A Statistical Perspective on Knowledge Discovery in Databases*, pages 83–116. In (Fayyad et al., 1996a).
- Farquhar, A., Fikes, R., and Rice, J. (1996). The Ontolingua Server: a Tool for Collaborative Ontology Construction. Tenth Knowledge Acquisition for Knowledge-Based Systems Workshop.
- Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors (1996a). *Advances in Knowledge Discovery and Data Mining*. AAAI Press / The MIT Press, Menlo Park, CA.
- Fayyad, U. M. (1996). Data Mining and Knowledge Discovery: Making Sense Out of Data. *IEEE Expert*, pages 20–25.
- Fayyad, U. M., Piatestsky-Shapiro, G., and Smyth, P. (1996b). *From Data Mining to Knowledge Discovery: An Overview*, pages 1–34. In (Fayyad et al., 1996a).
- Fayyad, U. M., Piatestsky-Shapiro, G., and Smyth, P. (1996c). The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM*, 39(11).

- Finin, T., Labrou, Y., and Mayfield, J. (1997). *KQML as an Agent Communication Language*, pages 291–316. In (Bradshaw, 1997b).
- Franklin, S. and Graesser, A. (1997). Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. pages 21–35. Third International Workshop on Agent Theories, Springer-Verlag.
- Gardner, S. R. (1998). Building the Data Warehouse. *Communications of the ACM*, 41(9):52–60.
- Genesereth, M. R. and Fikes, R. E. (1992). Knowledge Interchange Format. Technical Report 94305, Stanford University, Stanford, CA.
- Glymour, C., Mandigan, D., Pregibon, D., and Smyth, P. (1997). Statistical themes and lessons for data mining. *Data Mining and Knowledge Discovery*, 1(1):11–28.
- Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.
- Gray, R. S. (1997). Agent Tcl. *Dr. Dobbs's Journal*, pages 18–26.
- Hammer, J., Garcia-Molina, H., Widom, J., Labio, W., and Y.Zhuge (1995). The Stanford Data Warehousing Project. *IEEE Data Engineering Bulletin*, pages 41–48.
- Hendler, J. A. (1996). Intelligent Agents: Where AI Meets Information Technology. *IEEE Expert*, pages 20–23.
- Inmon, W. H. (1996). The Data Warehouse and Data Mining. *Communications of ACM*, 39(11):49–50.
- Jensen, D., Dong, Y., Lerner, B., McCall, E., Osterweil, L., Jr, S., and Wise, A. (1999). Coordinating Agent Activities in Knowledge Discovery Process. pages 137–146, San Francisco, CA. Int. Joint Conf. on Work Activities Coordination and Collaboration.
- Kargupta, H., Hamzaoglu, I., Sta, B., and Scalable, O. (1997). Distributed data mining using an agent based architecture. In *Knowledge Discovery And Data Mining*, pages 211–214, Menlo Park, CA. AAAI Press / The MIT Press.

- Kargupta, H., Park, B., Hershberger, D., and Johnson, E. (1999). *Collective Data Mining: A New Perspective Toward Distributed Data Mining*. AAAI Press / The MIT Press.
- Knoblock, C., Arens, Y., and Hsu, C. (1994). Cooperating Agents for Information Retrieval. 2nd International Conference on Cooperative Information Systems, University of Toronto Press.
- Knoblock, C. A. and Ambite, J. L. (1997). *Agents for Information Gathering*, pages 347–373. In (Bradshaw, 1997b).
- Kohavi, R., John, G., Long, R., Manley, D., and Pfleger, K. (1994). *MLC++: A Machine Learning Library in C++*. IEEE Computer Society Press.
- Lange, D. B. and Oshima, M. (1998). *Programming and Developing Java Mobile Agents with Aglets*. Addison-Wesley, Menlo Park, CA.
- Lôbo, R. B., Bezerra, L. A. F., and Oliveira, H. N. (1999). Avaliação Genética de Animais Jovens, Touros e Matrizes. Technical Report 71, GEMAC-FMRP-USP, Ribeirão Preto.
- Maes, P. (1994a). Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40.
- Maes, P. (1994b). Modeling Adaptive Autonomous Agents. *Artificial Life Journal*, 1(1):135–162.
- Mannila, H. (1996). Data Mining: Machine Learning, Statistic and Databases. pages 1–8. Eight International Conference on Scientific and Statistical Database Management.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.
- Muller, J. P., Wooldridge, M. J., and Jennings, N. R., editors (1997). *Intelligent Agents III. Agent Theories, Architecture and Languages*, volume 1193 of *Lecture Notes in Artificial Intelligence*, Budapest, Hungary. ECAI96 - Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.
- Neches, R. (1992). An Overview of the Darpa Knowledge Sharing Effort. In *Principles Of Knowledge Representation And Reasoning: Proceedings of the Third International Conference (KR92)*.

- Nodine, M., Perry, B., and Unruh, A. (1998). Experience with the InfoSleuth Agent Architecture. AAI-98 Workshop on Software Tools for Developing Agents.
- Nwana, H. S. (1996). Software Agents: An Overview. *Knowledge Engineering Review*, 11(3):205–244.
- Oates, T., Prasad, M., and Lesser, V. (1994). Networked Information Retrieval as Distributed Problem Solving. In *Proceedings of the CIKM '94 Workshop on Intelligent Information Agents*.
- Oliveira, R. B. T. and Rezende, S. O. (1998). Ferramentas de Visualização de Dados do MineSet. Technical Report 71, ICMC-USP.
- Oliviera, R. and Rezende, S. (2001). Minador: A Multi-agent System to Support the KDD Process. Innsbruck, Austria. Nineteenth IASTED International Conference Applied Informatics (AI 2001).
- Olofson, C. (1998). Oracle8i enables the third paradigm - cybersmart dbms for the enterprise. Technical Report IDC 15876, IDC, Framingham-MA.
- Pazzani, M. and Kibler, D. (1992). The Utility of Knowledge in Inductive Learning. *Machine Learning*, (9):57–94.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., Los Altos, California, USA.
- Rezende, S., Oliveira, R., Imamura, C., and Gonçalves, L. (2000). Text Mining em Documentos de Patentes usando o Sistema Minador. Rio de Janeiro. Anais do 21 Iberian Latin American Congress on Computational Methods in Engineering CILAMCE2000.
- Rezende, S. O., Monard, M. C., Rocha, C. A. J., Oliveira, R. B. T., Nagai, W., and Caulkins, C. W. (1998a). Extração de Conhecimento da Base de Dados do PMRGN. In *Avaliação Genética de Animais jovens, touros e matrizes*, pages 58–59. FMRP-USP/GEMAC.
- Rezende, S. O., Oliveira, R. B. T., Félix, L. C. M., and Rocha, C. A. J. (1998b). *Visualization for Knowledge Discovery in Database*, pages 81–95. In (Ebecken, 1998).

- Rezende, S. O. and Pugliesi, J. B. (1998). Aquisição de conhecimento explícito ou manual. Technical Report 37, ICMC-USP.
- Rocha, C. A. J. (1999). Redes Bayesianas para Extração de Conhecimento de Bases de Dados, Considerando a Incorporação de Conhecimento de Fundo e o Tratamento de Dados Incompletos. Dissertação de Mestrado, ICMC-USP.
- Russel, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewoods Cliff, NJ.
- Samos, J., Saltor, F., Sistac, J., and Bard'es, A. (1998). Database Architecture for Data Warehousing: An Evolutionary Approach. In Quirchmayr, G., Schweighofer, E., and Bench-Capon, T. J. M., editors, *Database and Expert Systems Applications*, number 1460 in Lecture Notes in Computer Science, pages 746–766, Vienna, Austria. 9th International Conference , DEXA'98, Springer-Verlag.
- Sen, A. and Jacob, V. S. (1998). Industrial-Strength Data Warehousing. *Communications of the ACM*, 41(9):29–31.
- Shoham, Y. (1997). *An Overview of Agent Oriented Programming*, pages 271–290. In (Bradshaw, 1997b).
- Silberschatz, A., Stonebraker, M., and Ullman, J. (1995). Database Research Achievments and Opportunities Into the 21st Century. *Workshop on the Futere of Database System Research*.
- Stolfo, S., Prodromidis, A., Tselepis, S., Lee, W., Fan, D., and Chan, P. (1997). JAM: Java Agents for Meta-learning over Distributed Databases. In *3rd International Conference on Knowledge Discovery and Data Mining* , pages 74–81, Newport Beach, CA. AAAI Press / The MIT Press.
- Tecuci, G. and Kodratoff, Y. (1995). *Machine Learning and Knowledge Acquisition: Integrated Approaches*. Academic Press.
- Unruh, A., Martin, G., and Perry, B. (1998). Getting Only What You Want: Data Mining and Event Detection Using InfoSleuth Agents. Technical Report 113, Microeletronics and Computer Technology Corporation (MCC), Austin, TX.

- Uthurusamy, R. (1996). *From Data Mining to Knowledge Discovery: Current Challenges and Future Directions*, pages 561–569. In (Fayyad et al., 1996a).
- van de Velde, W. (1996). Mediation Agents in Multi-Party Decision Making: An Experiment. In *First International Conference on the Practical Application of Intelligent Agents and Multi-agent Technology (PAAM'96)*, pages 629–654, UK. Blackpool.
- Wiederhold, G. (1996). *On the Barrier and Future of Knowledge Discovery*, pages vii–xi. In (Fayyad et al., 1996a).
- Witten, I. and Frank, E. (1999). *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann.
- Wooldridge, M. and Jennings, N. (1998). Pitfalls of Agent-Oriented Development. pages 385–391. 2nd International Conference on Autonomous Agents (Agents'98).
- Wooldridge, M. and Jennings, N. R. (1994). Agent Theories, Architectures, and Languages: A Survey. In Wooldridge, M. J. and Jennings, N. R., editors, *Intelligent Agents*, volume 890 of *Lecture Notes in Artificial Intelligence*, pages 1–39, Amsterdam, The Netherlands. ECAI94 - Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag.
- Wu, M. and Buchmann, A. (1997). An Overview of Data Warehousing and OLAP Technology. *BTW'97, Ulm*.
- Yang, J., Honavar, V., Miller, L., and Wong, J. (1998). Intelligent Mobile Agents for Information Retrieval and Knowledge Discovery from Distributed Data and Knowledge Sources. In *IEEE Information Technology Conference*, Syracuse, NY. IEEE Press.
- Zanasi, A. (1998). *From Data to Text Mining: Some Applicative Examples*. In (Ebecken, 1998).