Universidade Federal de Santa Catarina
Programa de Pós-Graduação em
Engenharia de Produção

UM MODELO DE FORMALIZAÇÃO DO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

Maria Madalena Dias

Tese apresentada ao Programa de
Pós-Graduação em Engenharia de Produção da
Universidade Federal de Santa Catarina
como requisito parcial para obtenção do título de
Doutor em Engenharia de Produção

Florianópolis
2001

Maria Madalena Dias

UM MODELO DE FORMALIZAÇÃO DO PROCESSO DE DESENVOLVIMENTO DE SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

Esta tese foi julgada e aprovada para a obtenção do título de Doutor em Engenharia de Produção no Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina

Florianópolis, 23 de fevereiro de 2001.

Prof. Ricardo Miranda Barcia, Dr.
Coordenador do Curso

BANCA EXAMINADORA

_____
Prof. Roberto C.Santos Pacheco, Dr.
Orientador

_____
Prof. Alex Alves Freitas, Dr.

_____
Profa. Tânia F. Calvi Tait, Dra.

_____
Prof. José Leomar Todesco, Dr.

_____
Prof. Aran Tcholakian Morales, Dr.

Ao meu esposo, Nardênio
pelo apoio e paciência.
À minha filha Natasha.

Agradecimentos

À Universidade Federal de Santa Catarina.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior CAPES.

Ao orientador Prof. Dr. Roberto C. S. Pacheco, pelo acompanhamento e dedicação.

Aos professores do Curso de Pós-Graduação em Engenharia de Produção, pelos ensinamentos que deram a base teórica e prática indispensáveis à realização desta pesquisa.

Aos meus colegas de curso, pela compreensão e apoio, especialmente ao André Vinícius Castoldi.

A todos que direta ou indiretamente contribuíram para a realização desta pesquisa.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

Após a organização conseguir sanar seus problemas operacionais, surge a necessidade de sistemas para o suporte à tomada de decisão. A área de pesquisa de mineração de dados cresce rapidamente para atender a essas novas necessidades. No entanto, a utilização de técnicas de mineração de dados torna-se difícil pela falta de uma metodologia completa e sistemática para o desenvolvimento de sistemas de descoberta de conhecimento. Esta tese apresenta um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, que inclui uma metodologia sistemática e rigorosa e um ambiente interativo para a implementação desses sistemas. A metodologia proposta integra UML (*Unified Modeling Language*) e Linguagem E-LOTOS (*Enhancements to Language Of Temporal Ordering Specification*). O principal objetivo é gerar informações relevantes à tomada de decisão, através da aplicação de técnicas de mineração de dados. O ambiente de implementação é baseado na Tecnologia de Agentes para facilitar o desempenho de suas tarefas. O modelo proposto foi aplicado na plataforma de informações da pós-graduação Brasileira (dados da CAPES 1998). Esta aplicação teve como principal objetivo validar a metodologia proposta, segundo critérios de análise encontrados na literatura.

Palavras-chave: Mineração de dados, sistemas de descoberta de conhecimento em banco de dados, métodos formais, modelagem orientada a objetos, tecnologia de agentes.

Abstract

After an organization having solved its operational problems, the need of systems appears for the support to the decision taking. Data mining is an area that is growing quickly to assist such new needs of the organization. However, the use of data mining techniques becomes difficult by the lack of a methodology that defines a complete and systematic process for the development of knowledge discovery systems. This thesis introduces a formalization model of development process of knowledge discovery systems in databases, including a rigorous and systematic methodology and an implementation environment for these systems. The methodology integrates UML (Unified Modeling Language) and E-LOTOS (Enhancements to Language Of Temporal Ordering Specification). The main objective is to generate relevant information to decision taking, with the application of data mining techniques. The environment is based on Agent Technology to facilitate the performance of its tasks. The model proposed was applied on Brazilian pos-graduation information (CAPES data 1998). The main objective of this application was to valid the methodological purpose, according to the criteria of analyzes found of the literature.

Keywords: Data mining, knowledge discovery in database, formal methods, object oriented, agent technology.

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Nos últimos anos, o uso de computadores passou a difundir-se pelos mais variados ramos de atividade e não ficou restrito apenas ao meio tecnológico. Os computadores e os sistemas de comunicação são responsáveis pela mudança na natureza de trabalho das pessoas e, além disso, estão reformulando o mundo dos negócios.

O conceito tradicional de trabalho, envolvendo grandes quantidades de papéis e relatórios complicados, tomada de decisão baseada em fatos pouco concretos, está sendo substituído pelo conceito moderno a que um sistema de informação está ligado. Este conceito reúne a utilização das mais variadas tecnologias para o ato de informar somente o que for relevante e de forma coerente e direta.

Durante alguns anos, a maioria das empresas acumulou muitas informações em seus bancos de dados, mas essas empresas, quase sempre, desconhecem o quanto essas informações podem ser úteis na busca de melhores perspectivas futuras para seus negócios.

Atualmente, existem ferramentas de *software* para o suporte à tomada de decisão que facilitam a geração de consultas em banco de dados e/ou permitem formas de análise de dados mais complexas. Essas ferramentas incluem sistemas OLAP (*On-Line Analytic Processing*) (Harrison, 1998).

As técnicas de mineração de dados, consideradas um dos tipos mais complexos de função analítica, surgiram com o intuito de revelar as informações estratégicas escondidas em grandes bancos de dados, através da pesquisa dessas informações e da determinação de padrões, classificações e associações entre elas. Essas informações valiosas podem ser utilizadas em muitas áreas, tais como (Cratochvil, 1999): marketing, instituições governamentais, saúde e finança.

A mineração de dados pode ser vista como uma parte fundamental do processo de descoberta de conhecimento em banco de dados.

Apesar da existência de ferramentas que auxiliam na tarefa de mineração de dados, ainda é sentida a carência de ferramentas para o desenvolvimento de sistemas de

descoberta de conhecimento em banco de dados que atendam às necessidades específicas de cada organização.

Outra técnica que tem sido reconhecida como bastante eficaz no desenvolvimento e gerenciamento de sistemas de descoberta de conhecimento em banco de dados é a tecnologia de agentes, principalmente quando se trata de extrair conhecimento em grandes bancos de dados localizados em nós diferentes de redes de computadores (Knapik e Johnson, 1998).

O desenvolvimento de sistemas de descoberta de conhecimento em banco de dados ainda se baseia em métodos '*ad-hoc*', devido à falta de uma metodologia completa e adequada que garanta a confiabilidade e a qualidade desses sistemas. Assim, melhorias são necessárias, tanto no aspecto formal de desenvolvimento desses sistemas, quanto em sua eficácia e sua eficiência.

1.2 OBJETIVOS

O objetivo geral desta tese de doutorado é definir um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Esse modelo engloba uma metodologia formal, denominada MeDesC e um ambiente de implementação desses sistemas, denominado ADesC.

Os objetivos específicos são:

1. Definir, detalhadamente, as etapas da metodologia proposta;
2. Utilizar diagramas UML (*Unified Modeling Language*) para representar os objetos do sistema e suas interações;
3. Definir uma forma de mapeamento dos diagramas UML para uma linguagem de especificação formal, tornando possível verificação e validação do sistema;
4. Definir uma estrutura de agentes móveis e os tipos de serviços necessários em sua implementação;
5. Especificar um ambiente de implementação de sistemas de descoberta de conhecimento em banco de dados;
6. Implementar um protótipo para o ambiente ADesC;
7. Desenvolver uma aplicação prática utilizando a metodologia MeDesc e implementar esta aplicação no ambiente ADesC.

1.3 JUSTIFICATIVAS

As principais justificativas para esta pesquisa são as seguintes:

- As metodologias existentes, que definem um processo completo de desenvolvimento de sistemas computacionais, geralmente, atendem às necessidades da maioria desses sistemas. No entanto, essas metodologias são inadequadas ao desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, devido ao fato desses sistemas se diferenciarem de outros tipos de sistemas, principalmente pela característica de indeterminismo neles presentes.
- As outras metodologias propostas para o desenvolvimento de sistemas de descoberta de conhecimento em banco de dados - Metodologia de Klemettinen, Metodologia de Feldens e Modelo de Processo CRISP-DM (ver seção 2.2.8 do Capítulo 2) - não incluem formalismo em nenhuma de suas etapas.
- A maioria das ferramentas de mineração de dados disponíveis implementa tipos específicos dessas técnicas (Goebel e Gruenwald, 1999), o que torna imprescindível um ambiente que implemente diferentes técnicas de mineração de dados.
- Este trabalho reúne importantes áreas atuais de pesquisa (mineração de dados, modelagem orientada a objetos, métodos formais e agentes inteligentes) na busca de uma metodologia sistemática e de um ambiente interativo e eficaz que suportem os requisitos de sistemas de descoberta de conhecimento em banco de dados.

1.4 CONTRIBUIÇÕES

A principal contribuição deste trabalho é a definição de um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Outras contribuições importantes:

- A definição de uma metodologia que formaliza modelos UML com o propósito de tornar mais rigorosa a aplicação de modelagem orientada a objetos no processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados;
- A descrição de uma forma de mapeamento dos diagramas UML para a linguagem de especificação formal E-LOTOS;
- A especificação de um ambiente de implementação de sistemas de descoberta de conhecimento em banco de dados;

- O uso de técnicas de mineração de dados na descoberta de conhecimento em banco de dados, segundo método formal de especificação;
- A possibilidade de integração de diferentes bancos de dados;
- O emprego da tecnologia de agentes na construção de sistemas de descoberta de conhecimento em banco de dados.
- A utilização das seguintes tecnologias básicas: modelagem orientada a objetos (UML) e técnica de descrição formal (E-LOTOS) nas etapas da metodologia proposta (MeDesC) e a tecnologia de agentes inteligentes na especificação do ambiente proposto (ADesC).

1.5 RESULTADOS ESPERADOS

Os resultados que se pretende obter com este trabalho são:

- Um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados;
- A definição detalhada da metodologia MeDesC;
- A especificação do ambiente ADesC;
- Um protótipo do ambiente ADesC;
- O desenvolvimento de uma aplicação prática utilizando a metodologia MeDesC e a implementação dessa aplicação através do protótipo do ambiente ADesC.

1.6 ORGANIZAÇÃO DO TRABALHO

Este trabalho, além deste capítulo que teve como fundamento situar e definir os objetivos do trabalho proposto, tem sua continuidade nos capítulos a seguir.

O Capítulo 2 (Fundamentação Teórica) apresenta um levantamento bibliográfico sobre mineração de dados, métodos formais e agentes inteligentes.

No Capítulo 3 (Metodologia de Desenvolvimento da Pesquisa) é descrita a metodologia de desenvolvimento desta pesquisa de tese de doutorado.

O Capítulo 4 (Modelo Geral Proposto e Metodologia de Desenvolvimento de Sistemas de Descoberta de Conhecimento em Banco de Dados) apresenta um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados e descreve as etapas da metodologia MeDesC.

O Capítulo 5 (Ambiente de Implementação de Sistemas de Descoberta de Conhecimento em Banco de Dados) apresenta a especificação completa do ambiente ADesC.

O Capítulo 6 (Aplicação do Modelo) apresenta a aplicação do modelo geral proposto através da especificação e da implementação de estudo de casos, tomando como base dados da CAPES, aplicando a metodologia MeDesC e utilizando o protótipo do ambiente ADesC.

Finalmente, no Capítulo 7 são apresentadas as conclusões deste trabalho e sugestões para pesquisas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

Durante várias décadas, desde a invenção do primeiro computador, o principal objetivo da utilização do computador é solucionar problemas operacionais da organização. A grande maioria das organizações ainda não possui meios de utilização dos recursos computacionais na tomada de decisão. Apesar da existência de grandes bancos de dados com muitas informações sobre o negócio da empresa, ainda são encontradas dificuldades na descoberta de conhecimento baseada nessas informações. Essas dificuldades podem estar relacionadas à falta de conhecimento da existência de técnicas de mineração de dados e/ou de ferramentas adequadas.

As técnicas de mineração de dados são aplicadas em sistemas de descoberta de conhecimento em banco de dados com o objetivo de extrair informações estratégicas escondidas em grandes bancos de dados, através da pesquisa dessas informações e da determinação de padrões, classificações e associações entre elas (Goebel e Gruenwald, 1999).

Os sistemas de descoberta de conhecimento são considerados sistemas complexos. Por isto, eles exigem maior rigor no seu processo de desenvolvimento.

Os métodos formais estão sendo atualmente muito utilizados na especificação de sistemas complexos com o objetivo de construir sistemas de forma mais sistemática e sem ambigüidades. Eles podem ser aplicados durante todo o processo de desenvolvimento de sistema ou apenas em determinadas fases do processo, com graus variados de rigor (Rushby, 1993).

Outra tecnologia avançada que pode ser utilizada na implementação de sistemas de descoberta de conhecimento em banco de dados é a tecnologia de agentes inteligentes, para facilitar, principalmente, as tarefas de busca e seleção de informações em grandes bancos de dados espalhados por redes de computadores, de escolha da técnica de mineração de dados mais adequada e de análise dos resultados.

Assim, o objetivo deste capítulo é apresentar uma fundamentação teórica sobre as áreas mais relevantes desta pesquisa, que são: mineração de dados, métodos formais e agentes inteligentes.

2.2 MINERAÇÃO DE DADOS

A mineração de dados pode ser considerada como uma parte do processo de Descoberta de Conhecimento em Banco de Dados (*KDD – Knowledge Discovery in Databases*).

Segundo Goebel e Gruenwald (1999), o termo KDD é usado para representar o processo de tornar dados de baixo nível em conhecimento de alto nível, enquanto mineração de dados pode ser definida como a extração de padrões ou modelos de dados observados.

A mineração de dados combina métodos e ferramentas das seguintes áreas: aprendizagem de máquina, estatística, banco de dados, sistemas especialistas e visualização de dados, conforme Figura 2.1 (Cratochvil, 1999).

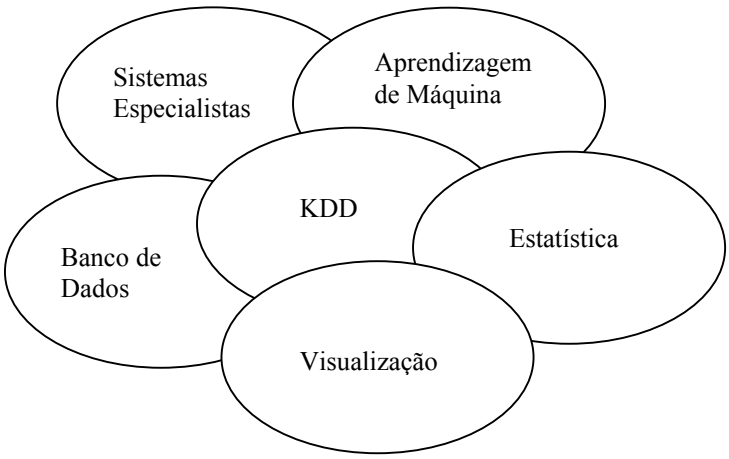


Figura 2.1: A Mineração de Dados como um Campo Multidisciplinar

2.2.1 Conceito de Mineração de dados

"Mineração de dados é a exploração e a análise, por meio automático ou semi-automático, de grandes quantidades de dados, a fim de descobrir padrões e regras significativos" (Berry e Linoff, 1997, p.5).

Os principais objetivos da mineração de dados são descobrir relacionamentos entre dados e fornecer subsídios para que possa ser feita uma previsão de tendências futuras baseada no passado.

Os resultados obtidos com a mineração de dados podem ser usados no gerenciamento de informação, processamento de pedidos de informação, tomada de decisão, controle de processo e muitas outras aplicações.

A mineração de dados pode ser aplicada de duas formas: como um processo de verificação e como um processo de descoberta (Groth, 1998). No processo de verificação, o usuário sugere uma hipótese acerca da relação entre os dados e tenta prová-la aplicando técnicas como análises estatística e multidimensional sobre um banco de dados contendo informações passadas. No processo de descoberta não é feita nenhuma suposição antecipada. Esse processo usa técnicas, tais como descoberta de regras de associação, árvores de decisão, algoritmos genéticos e redes neurais.

2.2.2 Origem dos Dados

As técnicas de mineração de dados podem ser aplicadas sobre bancos de dados operacionais ou sobre *Data Warehouse* (DW) ou *Data Mart*, nos quais geralmente resulta uma informação melhor, pois os dados normalmente são preparados antes de serem armazenados no DW ou *data mart* (Dias et al, 1998). Podem ser aplicadas, também, sobre um *data set*, que pode ser definido como um "banco de dados" (em um sentido fraco do termo) contendo apenas o conjunto de dados específico para um tipo de investigação a ser realizada.

"Um DW é um conjunto de dados baseado em assuntos, integrado, não-volátil e variante em relação ao tempo, de apoio às decisões gerenciais" (Inmon, 1997, p.33). No princípio, a expressão representava simplesmente um armazém de dados, como é a tradução de DW; porém, ao longo do tempo, vem recebendo diversos incrementos em sua estrutura.

Um DW tem por objetivo oferecer organização, gerenciamento e integração de bancos de dados, assim como ferramentas de exploração dos mesmos, para se obter vantagens competitivas no mercado. É construído tendo como base outros bancos de dados operacionais que podem estar implementados em diferentes plataformas na organização. É usado, geralmente, em aplicações de suporte à tomada de decisão.

Um *data mart* é um DW departamental, ou seja, um DW construído para uma área específica da organização (Inmon, 1997).

A técnica de *data mart* facilita a tomada de decisões em nível departamental e permite dados relacionais ou multidimensionais não voláteis (Dias et al, 1998).

2.2.3 Tarefas Desempenhadas por Técnicas de Mineração de dados

As técnicas de mineração de dados podem ser aplicadas a tarefas[1] como classificação, estimativa, associação, segmentação e sumarização. Essas tarefas são descritas a seguir.

a) Classificação

A tarefa de classificação consiste em construir um modelo de algum tipo que possa ser aplicado a dados não classificados visando categorizá-los em classes. Um objeto é examinado e classificado de acordo com uma classe definida (Harrison, 1998).

"A tarefa de classificação pode ser considerada uma tarefa mal definida, indeterminística, que é inevitável pelo fato de envolver predição" (Freitas, 2000, p. 65).

São exemplos de tarefas de classificação (Goebel e Gruenwald, 1999), (Mehta et al, 1996): classificar pedidos de créditos como de baixo, médio e alto risco; esclarecer pedidos de seguros fraudulentos; identificar a forma de tratamento na qual um paciente está mais propício a responder, baseando-se em classes de pacientes que respondem bem a determinado tipo de tratamento médico.

b) Estimativa (ou Regressão)

A estimativa é usada para definir um valor para alguma variável contínua desconhecida como, por exemplo, receita, altura ou saldo de cartão de crédito (Harrison, 1998). Ela lida com resultados contínuos, enquanto que a classificação lida com resultados discretos. Ela pode ser usada para executar uma tarefa de classificação, convencionando-se que diferentes faixas (intervalos) de valores contínuos correspondem a diferentes classes.

"Regressão é aprender uma função que mapea um item de dado para uma variável de predição real estimada" (Fayyad, 1996, p. 13).

Como exemplos de tarefas de estimativa tem-se (Fayyad, 1996), (Harrison, 1998): estimar o número de filhos em uma família; estimar a renda total de uma família; estimar o valor em tempo de vida de um cliente; estimar a probabilidade de que um paciente morrerá baseando-se nos resultados de um conjunto de diagnósticos médicos; prever a demanda de um consumidor para um novo produto.

[1] Neste contexto, tarefa é um tipo de problema de descoberta de conhecimento a ser solucionado.

c) Associação

A tarefa de associação consiste em determinar quais itens tendem a co-ocorrerem (serem adquiridos juntos) em uma mesma transação. O exemplo clássico é determinar quais produtos costumam ser colocados juntos em um carrinho de supermercado, daí o termo 'análise de *market basket*'. As cadeias de varejo usam associação para planejar a disposição dos produtos nas prateleiras das lojas ou em um catálogo, de modo que os itens geralmente adquiridos na mesma compra sejam vistos próximos entre si (Harrison, 1998).

"A tarefa de associação pode ser considerada uma tarefa bem definida, determinística e relativamente simples, que não envolve predição da mesma forma que a tarefa de classificação" (Freitas, 2000, p. 65).

d) Segmentação (ou *Clustering*)

A segmentação é um processo de partição de uma população heterogênea em vários subgrupos ou *clusters* mais homogêneos (Harrison, 1998). Na segmentação, não há classes predefinidas, os registros são agrupados de acordo com a semelhança, o que a diferencia da tarefa de classificação.

Exemplos de segmentação: agrupar os clientes por região do país, agrupar clientes com comportamento de compra similar (Goebel e Gruenwald, 1999); agrupar seções de usuários Web para prever comportamento futuro de usuário (Mobasher et al, 2000).

e) Sumarização

Segundo Fayyad (1996), a tarefa de sumarização envolve métodos para encontrar uma descrição compacta para um subconjunto de dados. Um simples exemplo desta tarefa poderia ser tabular o significado e desvios padrão para todos os itens de dados. Métodos mais sofisticados envolvem a derivação de regras de sumarização.

As tarefas de mineração de dados, descritas acima, são apresentadas de forma resumida na Tabela 2.1.

Tabela 2.1: Tarefas Realizadas por Técnicas de Mineração de Dados

TAREFA	DESCRIÇÃO	EXEMPLOS
Classificação	Constrói um modelo de algum tipo que possa ser aplicado a dados não classificados a fim de categorizá-los em classes	• Classificar pedidos de crédito • Esclarecer pedidos de seguros fraudulentos • Identificar a melhor forma de tratamento de um paciente
Estimativa (ou Regressão)	Usada para definir um valor para alguma variável contínua desconhecida	• Estimar o número de filhos ou a renda total de uma família • Estimar o valor em tempo de vida de um cliente • Estimar a probabilidade de que um paciente morrerá baseando-se nos resultados de diagnósticos médicos • Prever a demanda de um consumidor para um novo produto
Associação	Usada para determinar quais itens tendem a co-ocorrerem (serem adquiridos juntos) em uma mesma transação	• Determinar quais os produtos costumam ser colocados juntos em um carrinho de supermercado
Segmentação (ou *Clustering*)	Processo de partição de uma população heterogênea em vários subgrupos ou grupos mais homogêneos	• Agrupar clientes por região do país • Agrupar clientes com comportamento de compra similar • Agrupar seções de usuários Web para prever comportamento futuro de usuário
Sumarização	Envolve métodos para encontrar uma descrição compacta para um subconjunto de dados	• Tabular o significado e desvios padrão para todos os itens de dados • Derivar regras de síntese

2.2.4 Técnicas de Mineração de dados

Harrison (1998) afirma que não há uma técnica que resolva todos os problemas de mineração de dados. Diferentes métodos servem para diferentes propósitos, cada método oferece suas vantagens e suas desvantagens. A familiaridade com as técnicas é necessária para facilitar a escolha de uma delas de acordo com os problemas apresentados. A seguir são descritas as técnicas de mineração de dados normalmente usadas.

a) Descoberta de Regras de Associação

A técnica de descoberta de regras de associação estabelece uma correlação estatística entre certos itens de dados em um conjunto de dados (Goebel e Gruenwald, 1999).

Uma regra de associação tem a forma geral X1 ^ ... ^ Xn => Y [C,S], onde X1,..., Xn são itens que prevêem a ocorrência de Y com um grau de confiança C e com um suporte mínimo de S e "^" denota um operador de conjunção (AND). Um exemplo desta regra pode ser que 90% dos clientes que compram leite, também compram pão; o percentual de 90% é chamado a confiança da regra. O suporte da regra leite => pão é o número de ocorrências deste conjunto de itens na mesma transação.

A técnica de descoberta de regras de associação é apropriada à tarefa de associação.

Como exemplos de algoritmos que implementam regras de associação tem-se: Apriori, AprioriTid, AprioriHybrid, AIS, SETM (Agrawal e Srikant, 1994) e DHP (Chen et al, 1996).

b) Árvores de Decisão

Uma árvore de decisão é uma árvore onde cada nó não terminal representa um teste ou decisão sobre o item de dado considerado (Goebel e Gruenwald, 1999). O objetivo principal é separar as classes; tuplas de classes diferentes tendem a ser alocadas em subconjuntos diferentes, cada um descrito por regra simples em um ou mais itens de dados. Essas regras podem ser expressas como declarações lógicas, em uma linguagem como SQL, de modo que possam ser aplicadas diretamente a novas tuplas. Uma das vantagens principais das árvores de decisão é o fato de que o modelo é bem explicável, uma vez que tem a forma de regras explícitas (Harrison, 1998).

A técnica de árvore de decisão, em geral, é apropriada às seguintes tarefas: classificação e regressão.

Alguns exemplos de algoritmos de árvore de decisão são: CART, CHAID, C5.0, Quest (Two Crows, 1999), ID-3 (Chen et al, 1996), SLIQ (Metha et al, 1996) e SPRINT (Shafer et al, 1996).

c) Raciocínio Baseado em Casos

Também conhecido como MBR (*Memory-Based Reasoning* – raciocínio baseado em memória), o raciocínio baseado em casos tem base no método do vizinho mais próximo. "O MBR procura os vizinhos mais próximos nos exemplos conhecidos e combina seus valores para atribuir valores de classificação ou de previsão" (Harrison, 1998, p. 195). Tenta solucionar um dado problema fazendo uso direto de experiências e soluções passadas. A distância dos vizinhos dá uma medida da exatidão dos resultados.

Na aplicação do MBR, segundo Berry e Linoff (1997), existem quatro passos importantes: 1) escolher o conjunto de dados de treinamento; 2) determinar a função de distância; 3) escolher o número de vizinhos mais próximos; e 4) determinar a função de combinação.

A técnica de raciocínio baseado em casos é apropriada às seguintes tarefas: classificação e segmentação.

Os seguintes algoritmos implementam a técnica de raciocínio baseado em casos: BIRCH (Zhang et al, 1996), CLARANS (Chen et al, 1996) e CLIQUE (Agrawal et al, 1998).

d) Algoritmos Genéticos

Os algoritmos genéticos são métodos generalizados de busca e otimização que simulam os processos naturais de evolução. Um algoritmo genético é um procedimento iterativo para evoluir uma população de organismos e é usado em mineração de dados para formular hipóteses sobre dependências entre variáveis, na forma de algum formalismo interno (Goebel e Gruenwald, 1999).

Os algoritmos genéticos usam os operadores de seleção, cruzamento e mutação para desenvolver sucessivas gerações de soluções. Com a evolução do algoritmo, somente as soluções com maior poder de previsão sobrevivem, até os organismos convergirem em uma solução ideal (Harrison, 1998).

A técnica de algoritmos genéticos é apropriada às tarefas de classificação e segmentação.

Exemplos de algoritmos genéticos: Algoritmo Genético Simples (Goldberg, 1989), Genitor e CHC (Whitley, 1993), Algoritmo de Hillis (Hillis, 1997), GA-Nuggets (Freitas, 1999), GA-PVMINER (Araújo et al, 1999).

e) Redes Neurais Artificiais

As redes neurais são uma classe especial de sistemas modelados seguindo analogia com o funcionamento do cérebro humano e são formadas de neurônios artificiais conectados de maneira similar aos neurônios do cérebro humano (Goebel e Gruenwald, 1999).

"Como no cérebro humano, a intensidade de interconexões dos neurônios pode alterar (ou ser alterada por algoritmo de aprendizagem) em resposta a um estímulo ou uma saída obtida que permite a rede aprender" (Goebel e Gruenwald, 1999, p. 23).

Uma das principais vantagens das redes neurais é sua variedade de aplicação, mas os seus dados de entrada são difíceis de serem formados e os modelos produzidos por elas são difíceis de entender (Harrison, 1998).

A técnica de redes neurais é apropriada às seguintes tarefas: classificação, estimativa e segmentação.

Exemplos de redes neurais: Perceptron, Rede MLP, Redes de Kohonen, Rede Hopfield, Rede BAM, Redes ART, Rede IAC, Rede LVQ, Rede Counterpropagation, Rede RBF, Rede PNN, Rede Time Delay, Neocognitron, Rede BSB (Azevedo, 2000), (Braga, 2000), (Haykin, 2001).

A Tabela 2.2 apresenta um resumo das técnicas de mineração de dados aqui descritas.

Tabela 2.2: Técnicas de Mineração de Dados

TÉCNICA	DESCRIÇÃO	TAREFAS	EXEMPLOS
Descoberta de Regras de Associação	Estabelece uma correlação estatística entre atributos de dados e conjuntos de dados	• Associação	Apriori, AprioriTid, AprioriHybrid, AIS, SETM (Agrawal e Srikant, 1994) e DHP (Chen et al, 1996).
Árvores de Decisão	Hierarquização dos dados, baseada em estágios de decisão (nós) e na separação de classes e subconjuntos	• Classificação • Regressão	CART, CHAID, C5.0, Quest (Two Crows, 1999); ID-3 (Chen et al, 1996); SLIQ (Metha et al, 1996); SPRINT (Shafer et al, 1996).
Raciocínio Baseado em Casos ou MBR	Baseado no método do vizinho mais próximo, combina e compara atributos para estabelecer hierarquia de semelhança	• Classificação • Segmentação	BIRCH (Zhang et al, 1996); CLARANS (Chen et al, 1996); CLIQUE (Agrawal et al, 1998).
Algoritmos Genéticos	Métodos gerais de busca e otimização, inspirados na Teoria da Evolução, onde a cada nova geração, soluções melhores têm mais chance de ter "descendentes"	• Classificação • Segmentação	Algoritmo Genético Simples (Goldberg, 1989); Genitor, CHC (Whitley, 1993); Algoritmo de Hillis (Hillis, 1997); GA-Nuggets (Freitas, 1999); GA-PVMINER (Araújo et al, 1999).
Redes Neurais Artificiais	Modelos inspirados na fisiologia do cérebro, onde o conhecimento é fruto do mapa das conexões neuronais e dos pesos dessas conexões	• Classificação • Segmentação	Perceptron, Rede MLP, Redes de Kohonen, Rede Hopfield, Rede BAM, Redes ART, Rede IAC, Rede LVQ, Rede Counterpropagation, Rede RBF, Rede PNN, Rede Time Delay, Neocognitron, Rede BSB (Azevedo, 2000), (Braga, 2000), (Haykin, 2001)

2.2.5 Como Escolher a Técnica de Mineração de dados mais Adequada

A escolha de uma técnica de mineração de dados a ser aplicada não é uma tarefa fácil. Segundo Harrison (1998), a escolha das técnicas de mineração de dados dependerá da tarefa específica a ser executada e dos dados disponíveis para análise. Harrison (1998) sugere que a seleção das técnicas de mineração de dados deve ser dividida em dois passos: 1) traduzir o problema de negócio a ser resolvido em séries de tarefas de mineração de dados; 2) compreender a natureza dos dados disponíveis em termos de conteúdo e tipos de campos de dados e estrutura das relações entre os registros.

Essa escolha pode ser baseada, também, em critérios para classificação das técnicas. Uma relação desses tipos de critérios é dada por Harrison (1998).

"Diferentes esquemas de classificação podem ser usados para categorizar métodos de mineração de dados sobre os tipos de bancos de dados a serem estudados, os tipos de conhecimento a serem descobertos e os tipos de técnicas a serem utilizadas" (Chen et al,1996, p.4), como pode ser visto a seguir:

- Com que tipos de bancos de dados trabalhar:
  Um sistema de descoberta de conhecimento pode ser classificado de acordo com os tipos de bancos de dados sobre os quais técnicas de mineração de dados são aplicadas, tais como: bancos de dados relacionais, bancos de dados de transação, orientados a objetos, dedutivos, espaciais, temporais, de multimídia, heterogêneos, ativos, de herança, banco de informação de Internet e bases textuais.

- Qual o tipo de conhecimento a ser explorado:
  Vários tipos de conhecimento podem ser descobertos por extração de dados, incluindo regras de associação, regras características, regras de classificação, regras discriminantes, grupamento, evolução e análise de desvio.

- Qual tipo de técnica a ser utilizada:
  A extração de dados pode ser categorizada de acordo com as técnicas de mineração de dados subordinadas. Por exemplo, extração dirigida a dados, extração dirigida a questionamento e extração de dados interativa. Pode ser categorizada, também, de acordo com a abordagem de mineração de dados subordinada, tal como: extração de dados baseada em generalização, baseada em padrões, baseada em teorias estatísticas ou matemáticas, abordagens integradas, etc.

Atualmente, a descoberta de regras de associação parece ser uma das técnicas de mineração de dados mais utilizada, sendo encontrada em diversas pesquisas (Agrawal e Srikant, 1994), (Chen et al, 1996), (Hipp et al, 2000), (Holsheimer et al, 1996), (Mannila, 1997), (Viveros et al, 1996).

2.2.6 Áreas de Aplicação de Técnicas de Mineração de dados

A seguir, são relacionadas as principais áreas de interesse na utilização de mineração de dados, de acordo com Cratochvil (1999), (Mannila, 1996), (Viveros et al, 1996):

- **Marketing**. Técnicas de mineração de dados são aplicadas para descobrir preferências do consumidor e padrões de compra, com o objetivo de realizar marketing direto de produtos e ofertas promocionais, de acordo com o perfil do consumidor.
- **Detecção de fraudes**. Muitas fraudes óbvias (tais como, a compensação de cheque por pessoas falecidas) podem ser encontradas sem mineração de dados, mas padrões mais sutis de fraude podem ser difíceis de serem detectados, por exemplo, o desenvolvimento de modelos que predizem quem será um bom cliente ou aquele que poderá se tornar inadimplente em seus pagamentos.
- **Medicina**: caracterizar comportamento de paciente para prever visitas, identificar terapias médicas de sucesso para diferentes doenças, buscar por padrões de novas doenças.
- **Instituições governamentais**: descoberta de padrões para melhorar as coletas de taxas ou descobrir fraudes.
- **Ciência**: técnicas de mineração de dados podem ajudar cientistas em suas pesquisas, por exemplo, encontrar padrões em estruturas moleculares, dados genéticos, mudanças globais de clima, oferecendo conclusões valiosas rapidamente.
- **Controle de processos e controle de qualidade**: auxiliar no planejamento estratégico de linhas de produção e buscar por padrões de condições físicas na embalagem e armazenamento de produtos.

- **Banco**: detectar padrões de uso de cartão de crédito fraudulento, identificar clientes "leais", determinar gastos com cartão de crédito por grupos de clientes, encontrar correlações escondidas entre diferentes indicadores financeiros.
- **Apólice de seguro**: análise de reivindicações – determinar quais procedimentos médicos são reivindicados juntos, prever quais clientes comprarão novas apólices, identificar padrões de comportamento de clientes perigosos, identificar comportamento fraudulento.
- **Transporte**: determinar as escalas de distribuição entre distribuidores, analisar padrões de carga.
- **C & T** (Ciência e Tecnologia): avaliar grupos de pesquisa do país (Gonçalves, 2000), (Romão, 1999).
- **Web**: existem muitas pesquisas direcionadas à aplicação de mineração de dados na Web, tais como: (Loh et al, 2000), (Kosala e Blockeel, 2000), (Ma et al, 2000), (Mobasher et al, 2000), (Sarawagi e Nagaralu, 2000), (Spiliopoulou, 2000).

2.2.7 O Processo de Descoberta de Conhecimento

O processo de descoberta de conhecimento é um método semi-automático complexo e iterativo (Mannila, 1996). De acordo com Groth (1998), ele pode ser dividido em cinco passos básicos: preparação de dados, definição de um estudo, construção de um modelo, entendimento do modelo e predição.

Para Lans (1997), existe um passo que antecede a preparação de dados, trata-se da definição de objetivos.

A Figura 2.2 representa o processo de descoberta de conhecimento.

A seguir, os passos básicos do processo de descoberta de conhecimento são descritos.

1) Definição de Objetivos

Neste passo, deve-se definir os objetivos de negócio que deverão ser alcançados com a mineração de dados e o que deverá ser feito com os seus resultados, como por exemplo: mudança de plano de marketing.

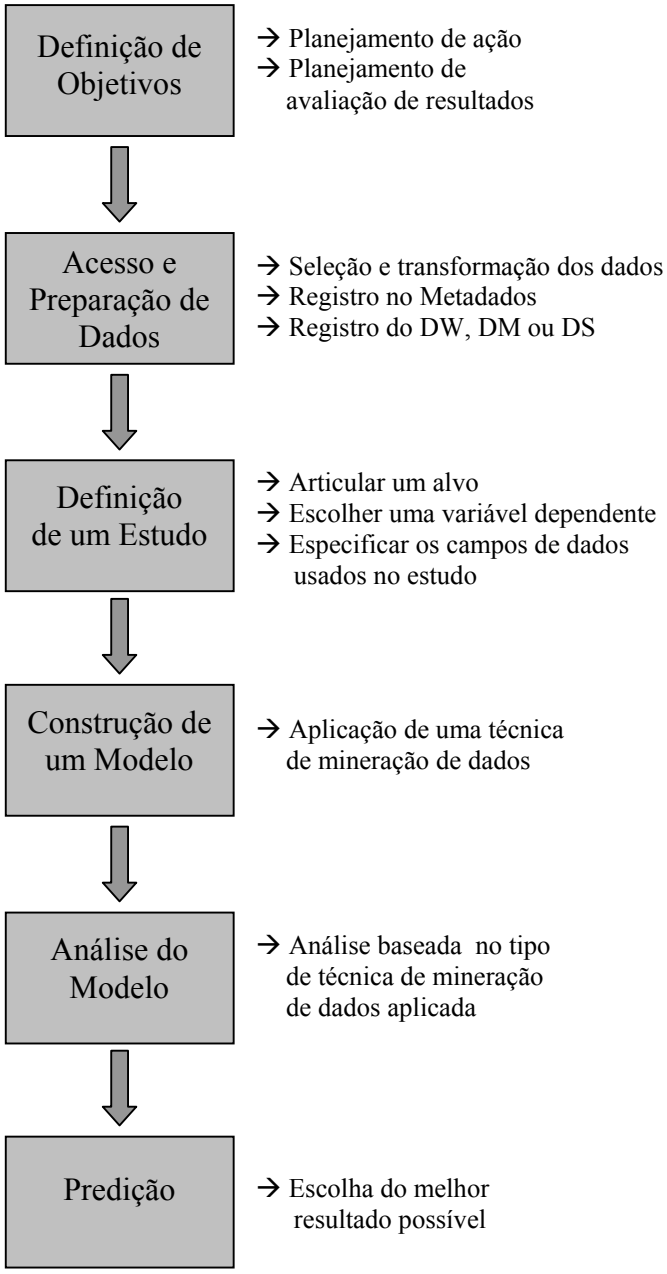


Figura 2.2: Processo de Descoberta de Conhecimento

2) Preparação de dados

A preparação de dados envolve as tarefas de seleção e transformação dos dados. Os tipos de dados selecionados podem estar organizados ao longo de múltiplas tabelas. Durante a seleção de dados, o usuário pode necessitar executar junções de tabelas ou eliminar linhas e/ou colunas de tabelas. Os métodos de transformação incluem organizar dados na forma desejada e converter um tipo de dado em outro tipo. A definição de

novos atributos é outro tipo de transformação que pode envolver aplicar operadores matemáticos ou lógicos sobre os valores de um ou mais atributos.

Os dados selecionados e transformados são armazenados em um DW, *data mart* ou *data set*. Para facilitar a realização desta fase, deve ser mantido um catálogo sobre as fontes de dados e sobre o que está no DW, *data mart* e *data set* no metadados. Os metadados são dados sobre as estruturas contidas em banco de dados. O metadados facilita o entendimento sobre o conteúdo e a estrutura de um DW, bem como a história das modificações realizadas.

A realização das tarefas deste passo exige conhecimento dos dados operacionais e de seus relacionamentos, disponibilidade de tempo do analista e/ou usuário e alguns cuidados na escolha de subconjuntos de atributos e de dados. Becher et al (2000) apresentam uma abordagem para a automatização desse passo e Hsu et al (2000) propõem o uso de um sistema semi-automático de limpeza de dados. Pyle (1999) descreve uma metodologia de preparação de dados.

3) Definição de um estudo

Definir um estudo pode envolver articular um alvo, escolher uma variável dependente ou uma saída que caracterize um aspecto do alvo e especificar os campos de dados que são usados no estudo. Bons estudos são unidos para caracterizar aquilo que pode ser descrito com seus dados. Por outro lado, o alvo pode ser usado para agrupar tipos similares de dados ou para identificar exceções em um conjunto de dados. A identificação de exceções é geralmente usada na descoberta de fraude ou de dados incorretos.

As atividades realizadas neste passo complementam os objetivos de negócio, definidos anteriormente, após a obtenção de um conhecimento mais detalhado sobre os dados operacionais existentes.

4) Construção de um modelo

A construção de um modelo é feita através de uma técnica de mineração de dados, tendo como base os dados transformados e o estudo definido no passo anterior. Um modelo resume grandes quantidades de dados por acumular indicadores. Alguns dos indicadores que vários modelos acumulam são: a) freqüências: mostram em qual freqüência que um certo valor ocorre; b) pesos ou impactos: indicam a influência exercida por algumas entradas na ocorrência de uma saída; c) conjunções: algumas

vezes certas entradas têm mais peso juntas do que separadas; d) diferenciação: indica a importância de uma entrada para uma determinada saída do que para uma outra saída.

5) Entendimento do modelo

Dependendo do tipo de modelo usado para representar os dados, existem diferentes formas de entendê-lo. Os indicadores que muitos modelos podem acumular, conforme descritos no passo anterior, podem influenciar no entendimento do modelo, além do tipo de técnica de mineração de dados aplicada na construção do modelo.

6) Predição

A predição é o processo de escolher o melhor resultado possível baseado na análise de dados históricos. O usuário deve analisar a informação descoberta de acordo com sua tarefa de suporte à decisão e objetivos. Portanto, ele precisa ter um bom entendimento sobre o negócio da empresa e sobre o conhecimento descoberto.

2.2.8 Metodologias para Sistemas de Descoberta de Conhecimento

O desenvolvimento de um sistema de descoberta de conhecimento em banco de dados é uma tarefa muito complexa, principalmente pela característica de indeterminismo deste tipo de sistema. Portanto, é imprescindível o uso de uma metodologia completa e sistemática.

> "Uma *metodologia de engenharia de software* é um processo para a produção organizada de software, com utilização de uma coleção de técnicas predefinidas e convenções de notação. Uma metodologia costuma ser apresentada como uma série de etapas, com técnicas e notação associadas a cada etapa." (Rumbaugh et al, 1994, p. 191).

Os trabalhos que se propõem apresentar uma metodologia para o desenvolvimento de sistemas de descoberta de conhecimento não incluem formalismo na especificação desses sistemas. Normalmente, as metodologias propostas procuram solucionar questões relativas a determinadas etapas do processo de desenvolvimento desses sistemas e não apresentam notação para representar as características do sistema como um todo.

A seguir, são relacionados três trabalhos que propõem uma metodologia para sistemas de descoberta de conhecimento em banco de dados.

a) Metodologia de Klemettinen

Klemettinen et al (1997) apresentam uma metodologia que pode ser usada para automatizar aquisição de conhecimento. As fases dessa metodologia são aquelas já definidas por outros autores (Fayyad, 1996), (Mannila, 1996): pré-processamento, transformação, descoberta, apresentação e utilização (ver Figura 2.3). No entanto, maior ênfase é dada nas duas fases centrais dessa metodologia:

- Fase de descoberta de padrões: onde são encontrados todos os padrões potencialmente relevantes para algum critério bastante livre;
- Fase de apresentação: onde são fornecidos métodos flexíveis para iterativa e interativamente criar diferentes visões para os padrões descobertos.

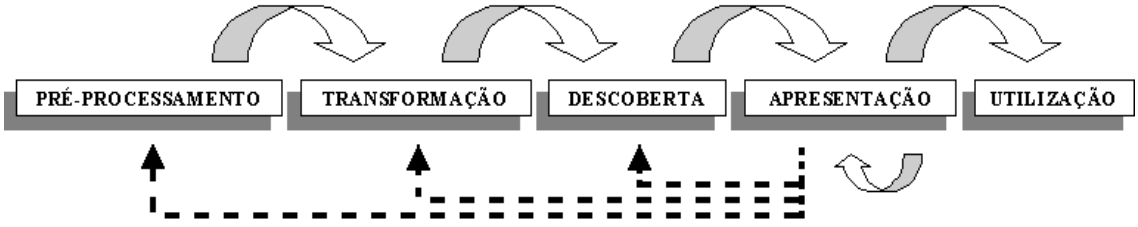


Figura 2.3: O Modelo do Processo KDD (Klemettinen et al, 1997)

Nas duas primeiras fases do processo, os dados são coletados e preparados de forma adequada para descoberta de padrões. Uma visão geral sobre os dados pode ser produzida nesta fase. Os atributos identificados como irrelevantes são removidos e novos atributos podem ser derivados.

Na fase de descoberta de padrões, todos os padrões potencialmente interessantes são gerados do conjunto do *data set*.

A apresentação do conhecimento descoberto é uma parte principal dessa metodologia. Nesta fase, os padrões relevantes podem ser localizados de grandes coleções de padrões potencialmente relevantes.

b) Metodologia de Feldens

Feldens et al (1998) propõem uma metodologia integrada, na qual as tecnologias de mineração de dados e *data warehouse*, bem como questões de visualização têm papéis muito importantes no processo. Também supõe uma forte interação entre mineradores de dados e pessoas da organização para questões de modelagem e preparação de dados. As fases definidas para esta metodologia são: pré-processamento, mineração de dados e pós-processamento, conforme Figura 2.4.

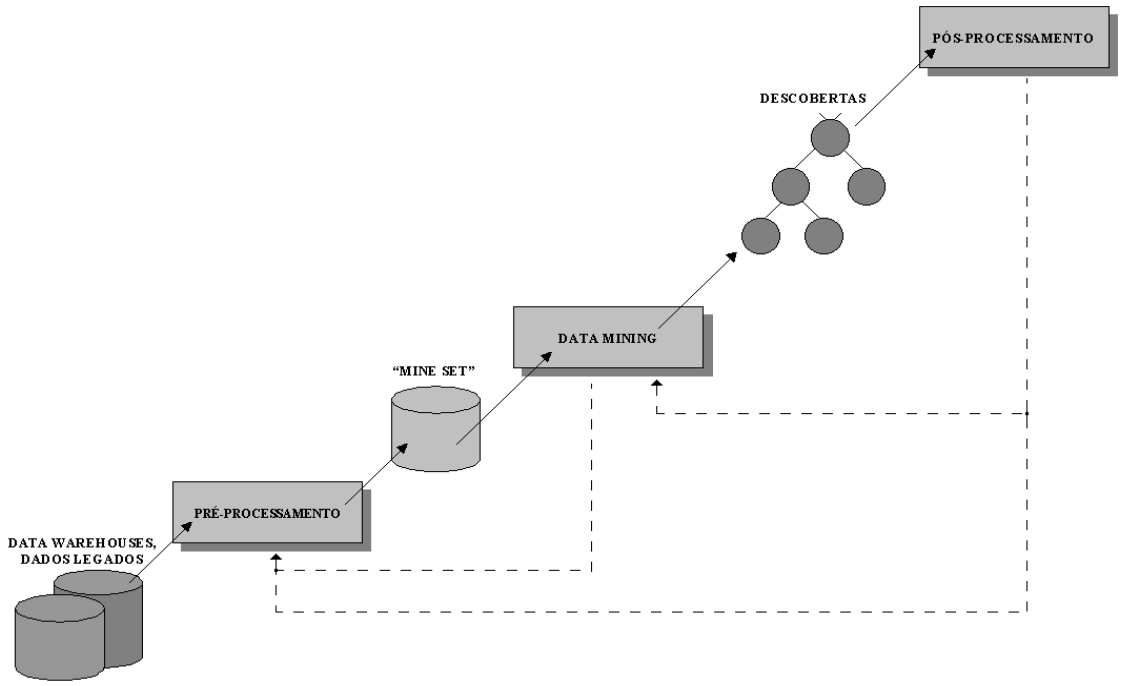


Figura 2.4: Processo KDD (Feldens et al, 1998)

A fase de pré-processamento inclui tudo o que é feito antes da mineração de dados, o que significa a análise que é feita na organização a fim de enfocar o projeto de mineração de dados, a análise dos dados existentes, integração de fontes de dados, transformações de dados, etc.

A fase de mineração de dados inclui a aplicação de algoritmos, possivelmente a aplicação repetida. A escolha dos algoritmos pode ser realizada baseando-se na análise que é feita na fase de pré-processamento.

A fase de pós-processamento pode ser definida por operações de filtragem, estruturação e classificação. Somente após esta fase, o conhecimento descoberto é

apresentado ao usuário. O conhecimento descoberto pode ser filtrado por alguma medida estatística, por exemplo, suporte, confiança ou outro critério definido pelo usuário. Estruturação significa que o conhecimento pode ser organizado de forma hierárquica.

c) Modelo de Processo CRISP-DM

O Modelo de Processo CRISP-DM (*CRoss-Industry Standard Process for Data Mining*) define um processo de mineração de dados não linear (CRIP-DM, 2001), conforme pode ser visto na Figura 2.5.

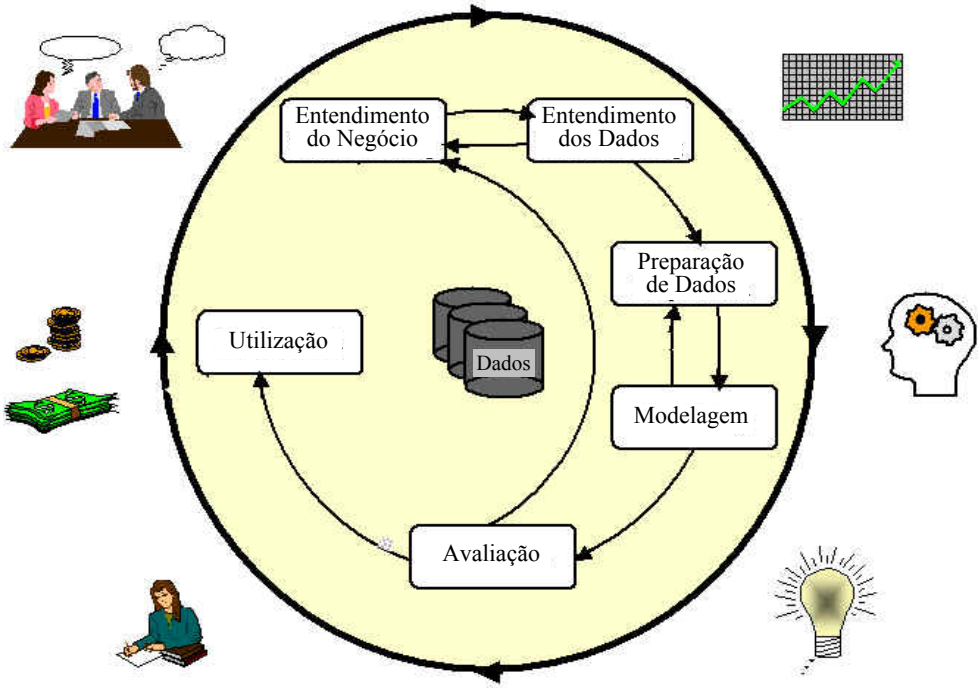


Figura 2.5: Fases do Modelo de Processo CRISP-DM (CRISP-DM, 2001)

Neste modelo, o ciclo de vida do projeto de mineração de dados consiste de seis fases. A seqüência dessas fases não é rigorosa, depende do resultado de cada fase ou de qual tarefa particular de uma fase precisa ser executada na próxima fase. As flechas indicam as dependências mais importantes e freqüentes entre as fases.

O círculo externo na figura simboliza a natureza cíclica da mineração de dados. Um processo de mineração de dados continua após uma solução ter sido descoberta. Os processos de mineração de dados subseqüentes se beneficiarão das experiências anteriores.

A seguir, cada fase do modelo é definida sucintamente.

A fase inicial do processo, Entendimento do Negócio (*Business Understanding*), visa o entendimento dos objetivos do projeto e dos requisitos sob o ponto de vista do negócio. Baseado no conhecimento adquirido, o problema de mineração de dados é definido e um plano preliminar é projetado para ativar os objetivos.

A fase Entendimento dos Dados (*Data Understanding*) inicia com uma coleção de dados e procede com atividades que visam: buscar familiaridade com os dados, identificar problemas de qualidade de dados, descobrir os primeiros discernimentos nos dados ou detectar subconjuntos interessantes para formar hipóteses da informação escondida.

A fase Preparação de Dados (*Data Preparation*) cobre todas as atividades de construção do *dataset* final. As tarefas de preparação de dados são, provavelmente, desempenhadas várias vezes e não em qualquer ordem prescrita. Estas tarefas incluem seleção de tabelas, registros e atributos, bem como transformação e limpeza dos dados para as ferramentas de modelagem.

Na fase Modelagem (*Modelling*), várias técnicas de modelagem são selecionadas e aplicadas e seus parâmetros são ajustados para valores ótimos. Geralmente, existem várias técnicas para o mesmo tipo de problema de mineração de dados. Algumas técnicas têm requisitos específicos na formação de dados. Portanto, retornar à fase de preparação de dados é freqüentemente necessário.

Na fase Avaliação (*Evaluation*), o modelo (ou modelos) construído na fase anterior é avaliado e são revistos os passos executados na sua construção para se ter certeza de que o modelo representa os objetivos do negócio. O principal objetivo é determinar se existe alguma questão de negócio importante que não foi suficientemente considerada. Nesta fase, uma decisão sobre o uso dos resultados de mineração de dados deverá ser alcançada.

Após o modelo (ou modelos) ser construído e avaliado, na fase Utilização, ou Aplicação, (*Deployment*) ele pode ser usado de duas formas. Na primeira forma, o analista pode recomendar ações a serem tomadas baseando-se simplesmente na visão do modelo e de seus resultados. Na segunda forma, o modelo pode ser aplicado a diferentes conjuntos de dados.

2.2.9 Ferramentas de Mineração de dados

De acordo com Goebel e Gruenwald (1999), muitas ferramentas atualmente disponíveis são ferramentas genéricas da Inteligência Artificial ou da comunidade de estatística. Tais ferramentas geralmente operam separadamente da fonte de dados, requerendo uma quantidade significativa de tempo gasto com exportação e importação de dados, pré- e pós-processamento e transformação de dados. Entretanto, segundo os autores, a conexão rígida entre a ferramenta de descoberta de conhecimento e a base de dados analisada, utilizando o suporte do SGBD (Sistema de Gerenciamento de Banco de Dados) existente, é claramente desejável. Para Goebel e Gruenwald (1999), as características a serem consideradas na escolha de uma ferramenta de descoberta de conhecimento devem ser as seguintes:

- A habilidade de acesso a uma variedade de fontes de dados, de forma *on-line* e *off-line*;
- A capacidade de incluir modelos de dados orientados a objetos ou modelos não padronizados (tal como multimídia, espacial ou temporal);
- A capacidade de processamento com relação ao número máximo de tabelas/tuplas/atributos;
- A capacidade de processamento com relação ao tamanho do banco de dados;
- Variedade de tipos de atributos que a ferramenta pode manipular; e
- Tipo de linguagem de consulta.

Existem ferramentas que implementam uma ou mais técnicas de mineração de dados. A Tabela 2.3 relaciona algumas dessas ferramentas, fornecendo informações tais como: a empresa fornecedora, as técnicas implementadas de mineração de dados e exemplos de aplicações.

Collier et al (1999) propõem uma metodologia para seleção de ferramentas de software de mineração de dados disponíveis no mercado.

Tabela 2.3: Ferramentas de Mineração de Dados

FERRAMENTA/ EMPRESA FORNECEDORA	TÉCNICAS DE MINERAÇÃO DE DADOS	APLICAÇÕES
AIRA/ Hycones IT (1998)	Regras de associação	Gerenciamento de relacionamento de cliente, marketing, detecção de fraude, controle de processo e controle de qualidade.
Alice 5.1/ Isoft AS. (1998)	Árvore de decisão Raciocínio baseado em casos	Política de crédito, marketing, saúde, controle de qualidade, recursos humanos.
Clementine/ Integral Solutions Limited (ISL, 1996)	Indução de regras Árvores de decisão Redes neurais	Marketing direto, identificação de oportunidades de venda cruzada, retenção de cliente, previsão de lucro do cliente, detecção de fraude, segmentação e lucro do cliente.
DataMind / DataMind Technology Center (1998), (Groth, 1998)	(abordagem própria)	Não identificadas.
Decision Series/ Neovista Solutions Inc. (1998)	Árvore de decisão Métodos estatísticos Indução de regras Redes neurais	Marketing direcionado, detecção de fraude, retenção de cliente, análise de risco, segmentação de cliente, análise de promoção.
Intelligent Miner/ IBM (1997)	Árvores de decisão Redes neurais	Segmentação de cliente, análise de conjunto de itens, detecção de fraude.
KnowledgeSEEKER/ Angoss IL (Groth, 1998)	Árvores de decisão Indução de regras	Lucro e segmentação de cliente para detecção de fraude e análise de risco, controle de processo, marketing direto.
MineSet/ Silicon Graphics Computer Systems (2000)	Métodos estatísticos Árvores de decisão Indução de regras	Áreas da saúde, farmacêutica, biotecnologia e química.
NeuralWorks Predict/ NeuralWare (Groth, 1998)	Rede neural	Indústria.
PolyAnalyst/ Megaputer Intelligence Ltd. (1998)	Algoritmo genético Métodos estatísticos Indução de regras	Marketing direto, pesquisa médica, análise de conjunto de itens.

2.3 MÉTODOS FORMAIS

"'Métodos Formais' são o uso de técnicas matemáticas no projeto e análise de *hardware* e *software* de computador; em particular, métodos formais permitem que propriedades de um sistema de computação sejam prognosticadas de um modelo matemático do sistema por um processo semelhante a cálculo" (Rushby, 1993, p. 7).

"Métodos formais são um conjunto de ferramentas e notações (com uma semântica formal) usado para especificar de forma não ambígua os requisitos de um

sistema que suporta a prova ou propriedades daquela especificação e provas de corretude de uma implementação para aquela especificação" (Wiryana, 1998, p. 16).

Segundo Sinnot e Turner (1994), um método formal é baseado em uma linguagem formal, isto é, uma notação simbólica que usa regras não ambíguas para desenvolver expressões legais naquela linguagem e para interpretar a semântica dessas expressões.

De acordo com Rushby (1993), as linguagens formais em ciência da computação, enriquecidas com algumas das idéias de linguagens de programação, são chamadas "linguagens de especificação", mas sua interpretação é geralmente baseada em uma lógica padrão.

Métodos Formais geralmente utilizam conceitos matemáticos e notações para definir precisamente teorias e modelos de comportamento da aplicação (Bates, 1996). Portanto, métodos formais modelam o comportamento discreto de sistemas computacionais usando matemática discreta.

De uma forma geral, no processo de desenvolvimento de sistemas, os requisitos e as especificações são documentados, geralmente, em linguagem natural; possivelmente com o auxílio de diagramas, equações, fluxogramas, dicionários de dados e pseudocódigo. Segundo Rushby (1993), o processo de desenvolvimento de sistemas considerados de segurança crítica e outros tipos de sistemas incluem componentes conhecidos como "verificação e validação" (V & V).

A verificação é o processo de determinar que cada nível de especificação e o próprio código final, completa e exclusivamente, implementam os requisitos da especificação como um todo. A verificação mostra que o sistema foi construído de acordo com os requisitos.

A validação, o outro componente de V & V, é o processo de confirmar que a especificação é uma representação verdadeira do mundo real. A validação mostra que os requisitos estão corretos.

Existem vantagens, dificuldades e custos associados com o uso de métodos formais. Segundo Rushby (1993), as vantagens variam com: a natureza e o fator crítico da aplicação; os estágios do ciclo de vida no qual métodos formais são usados; o grau de formalidade empregada; a qualidade do método e de quaisquer ferramentas mecanizadas que o suportam; e a experiência e competência dos profissionais.

> "Experiências mostram que a adição de métodos formais ao processo de desenvolvimento pode levar a um aumento significativo nos custos, mas que quando métodos formais são integrados completamente ao processo de desenvolvimento e os custos são medidos sobre todas as fases do ciclo de vida, os custos podem diminuir" (Liu et al, 1998, p. 24).

As vantagens exigidas a todos os métodos formais são a detecção de falhas o mais cedo possível. Isto ocorre porque as previsões podem ser realizadas o mais cedo possível no ciclo de vida e pelo fato dos métodos formais suportarem análises mais sistemáticas do que métodos informais.

A vantagem exigida para especificações formais é a redução da ambigüidade e da imprecisão de especificações em linguagem natural.

Segundo Wiryana (1998), o uso de métodos formais contribui para:

- A possibilidade de raciocínio sobre aspectos de usabilidade do sistema;
- A capacidade de execução da especificação do sistema;
- O refinamento e a estruturação do processo;
- A capacidade de modificação;
- A completeza da especificação.

2.3.1 Características de Métodos Formais

De acordo com Clarke e Wing (1997), alguns métodos formais, tais como Z, VDM e Larch, tratam da especificação do comportamento de sistemas seqüenciais. Nesses métodos, os estados são descritos em termos de estruturas matemáticas ricas como conjuntos, relações e funções; transições de estado são dadas em termos de pré- e pós-condições.

Clarke e Wing (1997) afirmam, também, que outros métodos, tais como CSP, CCS, Statecharts, Lógica Temporal e Autômato de E/S, tratam da especificação do comportamento de sistemas concorrentes; os estados variam tipicamente sobre domínios simples como inteiros e o comportamento é definido em termos de seqüências, árvores ou ordens parciais de eventos.

O uso dos conceitos matemáticos de abstração e composição é comum a todos esses métodos.

2.3.2 Classificação de Métodos Formais

Liu et al (1997) classificam métodos formais em cinco classes ou tipos:

a) Abordagem Baseada em Modelo;
b) Abordagem Baseada em Lógica;
c) Abordagem Algébrica;
d) Abordagem da Álgebra de Processos;
e) Abordagem Baseada em Rede.

a) Abordagem Baseada em Modelo

Nesta abordagem, um sistema é modelado explicitamente através da definição de estados e operações que o transformam de um estado a outro. Não existe representação explícita de concorrência. Exemplos:

- **Z**: uma especificação Z é basicamente composta como coleções ordenadas de definições de esquemas e descrições axiomáticas.
- **VDM** (*Vienna Development Method*): é similar ao Z na maioria dos aspectos e suporta composição e decomposição de modelo, que facilita muito a especificação de sistemas e a engenharia reversa.
- **Método-B**: usa a Notação de Máquina Abstrata para suportar a descrição do sistema e seu sucesso está no fato de já existir uma ferramenta *Toolkit B*.

b) Abordagem Baseada em Lógica

Esta abordagem usa lógicas para descrever propriedades desejadas do sistema, incluindo especificação de baixo nível, comportamentos temporal e probabilísticos. Exemplos:

- **ITL** (*Interval Temporal Logic*): é baseada em intervalos de tempo, idéia de como representar partes finitas de comportamento do sistema.
- **Cálculo de Duração**: é uma extensão da Lógica Temporal de Intervalo.
- **Lógica *Hoare***: fornece um meio de demonstrar que um programa é consistente com sua especificação; não é capaz de especificar um sistema de alto nível, entretanto, tem vantagens distintas nas especificações de baixo nível.

- **Cálculo WP** (*Weakest Precondition*): uma precondição descreve o estado inicial de um programa e uma pós-condição descreve o estado final.
- **Lógica Modal**: é o estudo de propriedades de contexto dependente, tais como necessidade e possibilidade. Em lógica modal, o significado de expressões depende de um contexto implítico, abstraído da linguagem objeto.
- **Lógica Temporal**: é usada para analisar a estrutura ou topologia de tempo.
- **RTTL** (*Real-Time Temporal Logic*): usa um domínio temporal distinto, as variáveis de estado ESM (*Extended State Machine*) e o conjunto de transições ESM para formar fórmula temporal.
- **RTL** (*Real-Time Logic*): é uma lógica de tempo-real com quatro conceitos básicos: ações, predicados de estado, eventos e restrições de tempo.
- **TPCTL** (*Timed Probabilistic Computation*): lida com restrições de tempo real e confiabilidade.

c) Abordagem Algébrica

Esta abordagem define explicitamente operações relatando o comportamento de diferentes operações sem definir estados. Exemplos:

- **OBJ**: é uma linguagem de especificação na qual uma álgebra é definida usando objetos.
- **LARCH**: um de seus objetivos é suportar uma variedade diferente de programação, incluindo linguagens imperativas, ao mesmo tempo localizar o máximo possível de dependências de linguagem.

d) Abordagem da Álgebra de Processos

Nesta abordagem, representação explícita e processos concorrentes são permitidos. O comportamento do sistema é representado por restrições sobre toda comunicação observável entre processos.

- **CSP** (*Communicating Sequential Process*): a notação de especificação formal CSP para sistemas concorrentes suporta um modelo de evento que permite a descrição de entidades que têm propriedades e relacionamentos que variam no tempo.

- **CCS** (*Calculus of Communicating Systems*): o modelo CCS consiste de um conjunto de processos comunicantes (agentes na terminologia CCS).
- **ACP** (*Algebra of Communicating Process*): é uma álgebra de processo baseada em ação, que pode ser vista como uma modificação de CCS.
- **LOTOS** (*Language Of Temporal Ordering Specification*): tem uma forte habilidade para descrever "dados" e "controle" do sistema, por fornecer um modelo comportamental derivado da álgebra de processo, principalmente de CCS e CSP, e permitir a especificação de tipos abstratos de dados e valores, baseados na linguagem ACT ONE.
- **TCSP** (*Timed* CSP): é uma extensão de CSP, com um denso modelo temporal fornecendo um relógio global.
- **TPCCS** (*Timed Probabilistic Calculus of Communicating Systems*): é essencialmente uma extensão de CCS, com tempo discreto e probabilidades.

e) Abordagem Baseada em Rede

Esta abordagem usa linguagens gráficas com uma semântica formal.

- **Rede de Petri**: a teoria de Rede de Petri é um dos primeiros formalismos para lidar com concorrência, não determinismo e conexões causais entre eventos, fornece uma representação gráfica com semântica formal de comportamento do sistema.
- **Rede de Petri Temporizada**: foi o primeiro formalismo concorrente para lidar com tempo real.
- **Statecharts**: fornecem um mecanismo de abstração baseado em máquina de estado finito, representa uma versão melhorada de métodos estruturados.

A Tabela 2.4 apresenta um resumo dessas abordagens.

Tabela 2.4: Classificação de Métodos Formais

ABORDAGEM	DESCRIÇÃO	EXEMPLOS
Baseada em Modelo	Modela o sistema através da definição de estados e operações que o transformam de um estado a outro	Z, VDM, Método-B.
Baseado em Lógica	Descreve propriedades do sistema, incluindo especificação de baixo nível, comportamentos temporal e probabilísticos	ITL, Cálculo de Duração, Lógica *Hoare*, Cálculo WP, Lógica Modal, Lógica Temporal, RTTL, RTL, TPCTL.
Algébrica	Define, explicitamente, operações relatando o comportamento de diferentes operações sem definir estados	OBJ, LARCH.
Álgebra de Processo	Representa o comportamento do sistema com restrições sobre toda comunicação observável entre processos	CSP, CCS, ACP, LOTOS, TCSP, TPCCS.
Baseada em Rede	Uso de linguagens gráficas com uma semântica formal	Rede de Petri, Rede de Petri Temporizada, Statecharts.

2.3.3 Níveis de Rigor em Métodos Formais

Os métodos formais podem ser aplicados em graus variados de rigor. Eles podem ser aplicados em estágios selecionados ou em todos os estágios do ciclo de vida do sistema, e para alguns ou todos os componentes e propriedades do sistema.

Rushby (1993) classifica métodos formais em quatro níveis de rigor crescente:

Nível 0: Nenhum uso de métodos formais.

Isto corresponde à prática padrão atual na qual a verificação é um processo manual de revisão e inspeção aplicado a documentos escritos em linguagem natural, pseudo-código, ou em uma linguagem de programação, possivelmente ampliada com diagramas. A validação é baseada em teste, que pode ser dirigido por requisitos e especificações (i.e., teste funcional ou de caixa-preta) ou pela estrutura do programa.

Nível 1: Uso de conceitos e notações da matemática discreta.

A idéia aqui é substituir alguma linguagem natural usada na declaração e especificação de requisitos com notações e conceitos derivados da matemática discreta e lógica. Os métodos formais de nível 1 geralmente ampliam um método de desenvolvimento existente.

Nível 2: Uso de linguagens de especificação formalizadas com algumas ferramentas de suporte mecanizadas.

Os métodos formais de nível 2 podem fornecer ferramentas mais simples, tais como checadores de sintaxe, que examinam somente as características superficiais de especificações; ou ferramentas mais complexas, tais como checadores de tipo, que examinam mais precisamente seu conteúdo.

Nível 3: Uso de linguagens de especificação totalmente formais, incluindo teorema mecanizado, provando ou checando provas.

Quando métodos de prova são completamente formalizados (i.e., reduzidos à manipulação de símbolo), torna-se possível mecanizá-los. A mecanização pode ter a forma de um checador de prova (i.e., um programa de computador que checa os passos de uma prova proposta pelo usuário) ou de um provador de teorema (i.e., um programa de computador que tenta provar sem intervenção humana) ou, mais comumente, alguma coisa entre eles.

Níveis mais altos de rigor na aplicação de métodos formais nem sempre são preferíveis ao invés de níveis de menor rigor. A escolha do nível de rigor depende dos benefícios desejados no uso de métodos formais, do fator de segurança crítica da aplicação e dos recursos disponíveis.

A inserção de um método formal em um processo de desenvolvimento parte da identificação do nível desejado.

2.3.4 Aplicação de Métodos Formais

Para Rushby (1993), assim como é possível variar o nível de rigor de métodos formais, também é possível variar a extensão de sua aplicação. Assim, os métodos formais podem ser aplicados:

- Para todas ou somente algumas das etapas do ciclo de vida de desenvolvimento de sistemas. Eles são aplicados, geralmente, nas primeiras etapas do ciclo de vida onde os métodos convencionais são aparentemente mais fracos.
- Para todos ou somente alguns componentes selecionados do sistema. Podendo variar o nível de sua aplicação para cada componente.

- Somente para propriedades do sistema consideradas como as mais importantes. Em um sistema de controle de tráfego aéreo, por exemplo, algumas vezes pode ser mais importante estar certo de que um componente não provoque certos tipos de falhas catastróficas do que ele tenha certas propriedades positivas.

2.3.5 Metodologias Incluindo Métodos Formais

A seguir são descritas metodologias que incluem métodos formais no processo de desenvolvimento de sistemas.

a) Método ROOA

Moreira e Clark (1993) propõem o método ROOA (*Rigorous Object-Oriented Analysis*) que introduz rigor no processo de análise orientada a objetos, oferecendo um conjunto de regras que permite produzir sistematicamente um modelo formal de análise orientada a objetos a partir dos requisitos originais que os métodos informais de análise orientada a objetos geralmente propõem.

A técnica de descrição formal escolhida foi LOTOS porque "LOTOS tem uma semântica matemática, precisa e formal e pode ser usada para escrever especificações num estilo orientado a objetos" (Moreira e Clark, 1993, p.139).

O método ROOA envolve três tarefas principais:

1) Construir o Modelo de Objetos;
2) Refinar o Modelo de Objetos;
3) Construir o Modelo Formal LOTOS.

A Figura 2.6 representa as tarefas do método ROOA.

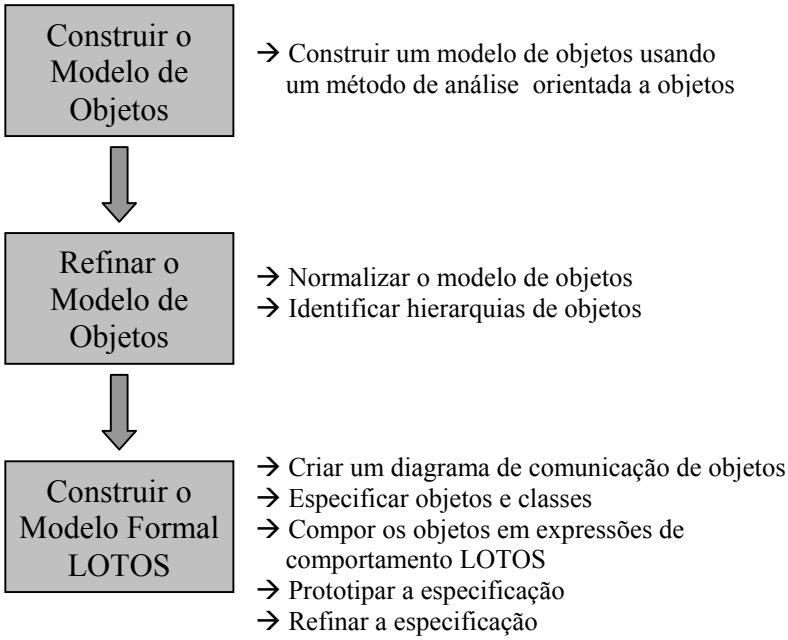


Figura 2.6: Tarefas do Método ROOA

b) Metodologia SOFL

Liu et al (1998) propõem uma linguagem chamada SOFL (*Structured-Object-based-Formal Language*) e a metodologia SOFL, que integra métodos estruturados baseados em diagramas de fluxo de dados estendidos e formalizados, metodologia orientada a objetos e notação formal VDM-SL.

A construção de um sistema, baseada nessa metodologia, usa métodos estruturados na análise de requisitos e especificações, uma metodologia orientada a objetos durante as fases de projeto e implementação, com métodos formais aplicados durante todo o desenvolvimento para fornecer especificações de alta qualidade e verificações de vários níveis do sistema.

O processo de desenvolvimento de sistema usando SOFL consiste de duas fases principais: *desenvolvimento estático e desenvolvimento dinâmico*. O desenvolvimento estático refere-se às atividades que melhoram o programa até que os requisitos dos usuários sejam satisfeitos.

O desenvolvimento estático é dividido em quatro estágios: análise preliminar de requisitos, especificação formal dos requisitos, projeto e implementação.

O desenvolvimento dinâmico é o processo de descobrir características dinâmicas do sistema, verificando se o sistema atende às necessidades atuais do usuário e adquirindo mais requisitos por meio de prototipação e teste.

O desenvolvimento estático e o dinâmico podem interagir-se na prática; isto é, quando o desenvolvimento estático atinge um certo nível, algumas partes já especificadas podem ser prototipadas em paralelo com o desenvolvimento estático restante.

A Figura 2.7 representa as fases da metodologia SOFL.

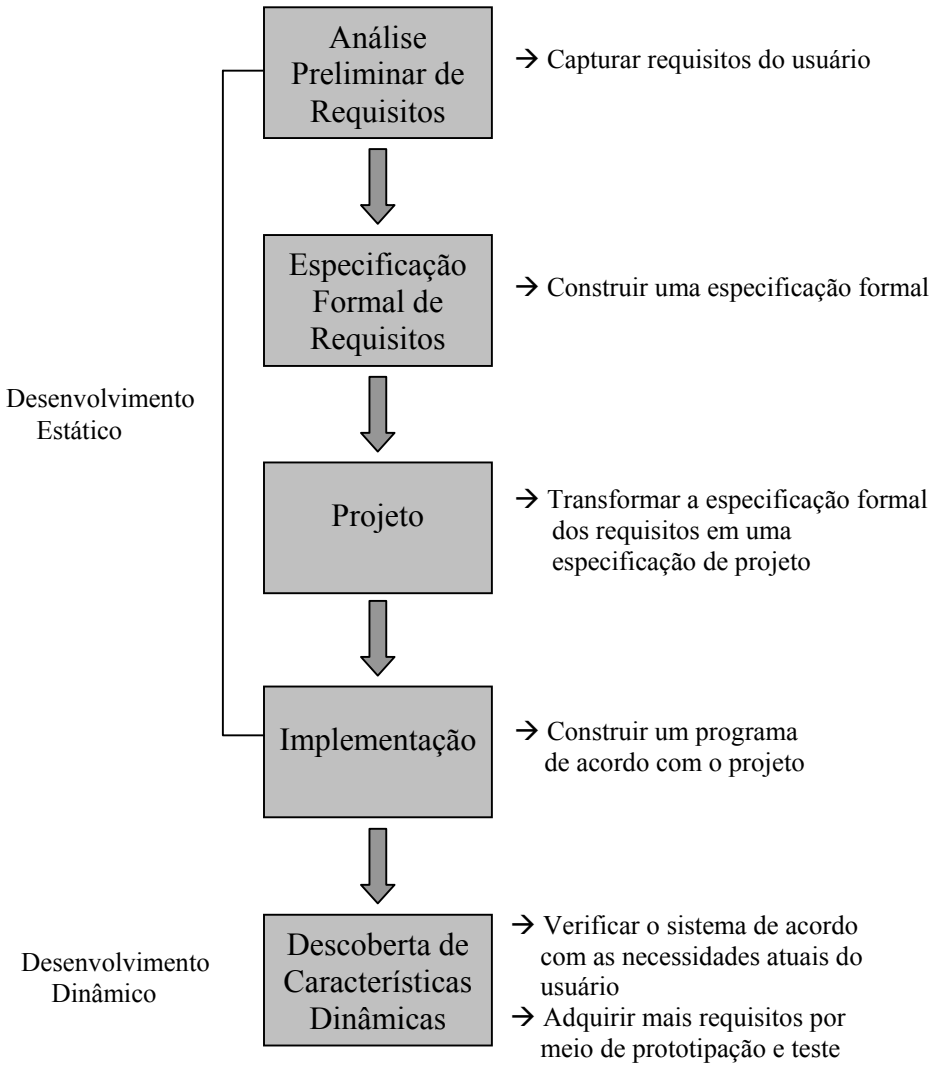


Figura 2.7: Fases da Metodologia SOFL

2.4 AGENTES INTELIGENTES

"Um agente pode ser definido como algo ou alguém que age representando outra parte, com propósito de desempenhar funções específicas em benefício da parte representada" (Heilmann et al, 1995, p. 5).

Desta forma, um programa computacional que realiza tarefas específicas para seu usuário pode ser chamado de agente de *software*.

Segundo Knapik e Johnson (1998), os maiores componentes de um sistema de agentes são: um construtor de agente ou ambiente de desenvolvimento adequado, uma infra-estrutura de agentes e os agentes, que juntos compõem uma arquitetura dentro de um ambiente de execução.

2.4.1 Características de Agentes

A Tabela 2.5 relaciona as principais características de agentes, de acordo com (Wooldridge e Jennings, 1995), (Heilmann et al, 1995), (Knapik e Johnson, 1998) e (Cheung e Chan, 2000):

Tabela 2.5: Características de Agentes

Característica	**Descrição**
Autonomia	Capacidade de operar sem nenhuma intervenção direta, tendo controle interno sobre suas ações e seus estados.
Inteligência	Refere-se à propriedade que habilita o agente a negociar efetivamente com ambigüidades.
Habilidade de comunicação/cooperação	Capacidade de interagir com outros agentes (ou seres humanos) via algum tipo de linguagem de comunicação de agentes.
Mobilidade	Capacidade de transportar-se de uma máquina a outra.
Reatividade	Capacidade de perceber seu ambiente e responder a mudanças que nele ocorrem.
Raciocínio	Capacidade de raciocínio implica que um agente pode possuir a habilidade de inferir e extrapolar baseando-se em conhecimento atual e experiências anteriores.
Comportamento adaptativo/Aprendizagem	Capacidade de examinar o ambiente externo (a Web, por exemplo) e adaptar suas ações baseando-se em ações tomadas anteriormente sob condições similares.

A inserção de Inteligência Artificial no trabalho deve-se às características de autonomia, inteligência, mobilidade e raciocínio, relevantes para a implementação das funções do ambiente ADesC (Capítulo 5).

2.4.2 Aplicação de Agentes

Os agentes podem ser aplicados no desempenho de muitas tarefas complementares ao homem, tais como (Knapik e Johnson, 1998):

- Consulta a bancos de dados. Esta tarefa é a principal candidata à intervenção de agentes e requer integrar grandes quantidades de dados. Um agente pode chegar em um servidor de banco de dados e fazer uma série de pedidos de informação, então analisar a coleção resultante e retornar somente a informação que o usuário realmente desejar. O acesso a banco de dados pode ser via Web.
- Gerenciamento de Redes Locais. Os agentes podem monitorar atividades em um sistema distribuído, tal como checar perturbação em uma rede local e fornecer relatórios para uso por *software* de gerenciamento de rede ou monitorar nós em uma rede.
- Aprendizagem de experiências dentro de um domínio restrito. Os agentes são adequados para o desenvolvimento de *software* de aprendizagem. Algoritmo genético e rede neural são boas abordagens para tornar um agente capaz de avaliar e aprender.
- Monitoramento e análise de cadeias de dados em tempo real. Os agentes de sistema embutido desempenham este tipo de atividade, pela impossibilidade de ser realizada por ser humano.
- Execução contínua de tarefas em *background* (último plano), tal como checar erros em um banco de dados ou condições ilegais (violações de regras) em um sistema.
- Disparo de tarefas repetitivas e periódicas baseando-se no tempo ou em condições do sistema.
- Desempenho de procedimentos de negócio para o qual o fluxo de trabalho é bem definido.
- Gerenciamento de recursos do sistema tal como memória e bancos de dados. Os agentes podem restringir o acesso e alertar gerenciamento de sistema sobre possíveis problemas.
- Extração e análise de informações de relacionamentos de dados complexos em sistemas de suporte à tomada de decisão.

- Manutenção da segurança em sistemas distribuídos.
- Gerenciamento de recursos do sistema telefônico.
- Extração de informações sobre legislação e regulamentos em bases de dados governamentais.
- Monitoramento e triagem de pacientes, assistência em diagnósticos, informação sobre condições do paciente.

2.4.3 Classificação de Agentes Baseada no Grau de Mobilidade

Os agentes podem ser classificados baseando-se no grau de mobilidade que eles possuem. Eles podem ser fixos ou estacionários ou então móveis e itinerantes.

Os agentes fixos ou estacionários executam tarefas independentes sobre um processador (por exemplo, cliente ou servidor), mas podem gerar outras tarefas para serem executadas sobre o mesmo ou outros processadores. Os agentes estacionários trabalham muito bem na busca de informação e como filtros, monitores de cadeia de dados, controladores de dispositivo inteligente, agentes de triagem de E-mail e informantes. Agentes fixos permanecem em uma única localização durante toda a sua execução. Um agente de interface é um tipo de agente fixo.

Uma das propriedades mais atrativas de um agente móvel é a sua capacidade de mover-se de um lugar para outro autonomamente. De acordo com Falsarella et al (1996), diferentes tipos ou graus de mobilidade podem existir:

- Estático: agente que só será executado na própria plataforma em que foi originalmente criado;
- Execução Remota: agente que é enviado para um nó remoto para executar suas tarefas. Este agente inicia sua execução no local remoto sempre a partir do seu ponto inicial;
- Migração: agente que pode parar sua execução em um nó, migrar para qualquer outro nó e continuar sua execução do ponto onde parou antes da migração. Para isso ele deve levar consigo seu estado. Entende-se por estado do agente todas as informações necessárias (valores de variáveis, próxima instrução a ser executada, valores de parâmetros etc.) para que o mesmo possa dar continuidade a sua tarefa no novo nó.

2.4.4 Agentes Móveis

De acordo com Nwana (1996), agentes móveis são processos computacionais de *software* capazes de navegar por redes de longa distância, tais como a WWW (*World Wide Web*), interagindo com outras estações, colhendo informações em nome de seu proprietário e retornando a sua fonte tendo desempenhado os deveres que lhes foram atribuídos.

Além das características gerais de agentes relacionadas anteriormente, agentes móveis possuem outras características importantes relacionadas à mobilidade. São elas (Lange e Chang, 1996):

- Passagem de objeto: quando um objeto se move, o objeto como um todo é passado; ou seja, seu código, seus dados, seu estado de execução e seu itinerário de viagem são passados juntos.
- Assincronismo: o agente móvel tem seu próprio procedimento de execução.
- Interação local: o agente móvel interage com outros agentes móveis ou objetos estacionários. Se necessário, ele pode enviar agentes mensageiros ou agentes substitutos, que são todos agentes móveis, para facilitar interação remota.
- Operação sem conexão: o agente móvel pode desempenhar suas tarefas quando a conexão da rede está aberta ou fechada. Se a conexão de rede está fechada e ele necessita mover-se, ele pode esperar até que a conexão seja reaberta.
- Execução paralela: mais de um agente móvel pode ser enviado para diferentes sítios para executar tarefas em paralelo.

Segundo Falsarella et al (1996), o uso de agentes móveis em certas aplicações pode ser justificado pelos benefícios e vantagens que podem trazer, tais como:

- Redução do custo de comunicação: dependendo da quantidade de informações a serem transmitidas pelo agente entre dois nós da rede, pode ser mais eficiente em termos de custo de comunicação enviar um agente para onde estas estão localizadas. Desta forma, o agente pode filtrar e selecionar somente as informações relevantes e as transferir para o seu nó de origem.

- Processamento assíncrono: um agente pode ser enviado através da rede, continuando suas tarefas em outros nós. Enquanto o agente se encontra fora de seu nó de origem não é necessário que este nó permaneça em operação.
- Divisão de carga de processamento: computadores com baixa capacidade de processamento podem ser usados eficientemente para armazenar e processar dados que são filtrados e enviados por agentes móveis a partir de grandes servidores de informações.
- Flexibilidade: agentes móveis fornecem uma única arquitetura de processamento distribuído que funciona de forma diferente de cargas estáticas. São uma alternativa para o tradicional modelo cliente-servidor.

Devido à complexidade existente na implementação de agentes móveis, é interessante a definição de uma estrutura de agentes. A seguir são relacionadas algumas estruturas de agentes móveis disponíveis:

a) Concordia

De acordo com a Mitsubishi Eletric Informatic Technology Center America (1997), Concordia é uma estrutura para o desenvolvimento e gerenciamento de aplicações de agentes móveis. Esta estrutura consiste de múltiplos componentes escritos totalmente na linguagem Java.

As aplicações de agentes móveis usam serviços Concordia para mover agentes sobre uma rede de máquinas distribuídas e para terem acesso a serviços disponíveis dessa rede. Exemplos comuns desses serviços são GUIs (*Graphic User Interfaces*), bases de dados e outros agentes.

b) Aglets

Segundo Lange e Chang (1998) e Lange (1998), Aglets da IBM são programas Java móveis que podem navegar e executar em nós da rede. Um aglet que executa em um nó pode suspender sua execução a qualquer momento, dirigir-se a um nó remoto e lá retomar sua execução.

c) CyberAgentes

CyberAgentes são, também, programas Java móveis desenvolvidos pela FTP *Software*. Calcote (1998) afirma que eles contêm tudo o que é necessário para construir,

lançar, receber e gerenciar agentes inteligentes. Especificamente, eles fornecem um ambiente de desenvolvimento integrado, um Gerente CyberAgent e agentes de amostra.

d) Tabriz

A Tabriz é uma ferramenta de desenvolvimento de aplicações da General Magic (1996), originada de uma adaptação da linguagem Telescript para a *World Wide Web*. Ela permite a criação de agentes não móveis que podem usar a *Web* para ter acesso a páginas de informação e processá-las localmente.

A Tabriz possui algumas limitações. Primeiro, ela é baseada em um servidor proprietário que somente executa em certos sistemas Unix. Segundo, ela é baseada na linguagem Telescript, que não tem um padrão divulgado e nem implementação publicamente disponível. Terceiro, ela reside totalmente em um único servidor.

e) Agent Tcl

De acordo com Gray (1998), Agent Tcl é um sistema de agente transportável, onde os agentes são escritos em uma versão estendida da linguagem TCL. Cada agente pode suspender sua execução a qualquer momento, transportar-se para uma outra máquina e reassumir a execução na nova máquina.

O sistema Agent Tcl é implementado como dois componentes. O primeiro componente é um interpretador Tcl estendido. O segundo componente é um servidor que executa sobre cada máquina. Uma versão alfa de Agent Tcl está disponível para uso público.

2.5 CONSIDERAÇÕES FINAIS

A mineração de dados surgiu com o objetivo principal de dar suporte à tomada de decisões na empresa. Portanto, a aplicação de técnicas de mineração de dados em sistemas de descoberta de conhecimento em banco de dados busca a descoberta de regras e padrões em dados que trarão o conhecimento suficiente e adequado para aquelas pessoas responsáveis pela tomada de decisões na empresa.

O usuário de um sistema de descoberta de conhecimento em banco de dados precisa ter um sólido entendimento do negócio da empresa para ser capaz de selecionar corretamente os subconjuntos de dados e as classes de padrões mais interessantes.

As técnicas de mineração de dados podem ser aplicadas sobre bancos de dados operacionais ou sobre *data warehouse*/*data mart*. Os bancos de dados operacionais podem estar espalhados por redes de computadores. Nesse caso, a tecnologia de agentes móveis pode ser aplicada no desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, por ter essa classe de agentes a capacidade e a autonomia de migração, o que facilita a busca e a seleção de informações. A característica de migração inerente a este tipo de agente é utilizada neste trabalho de tese na especificação do ambiente ADesC (Capítulo 5).

Nos últimos anos, houve um grande crescimento da utilização de abordagens orientadas a objetos no desenvolvimento de sistemas computacionais. Mas, apesar de metodologias orientadas a objetos apresentarem um método sistemático de desenvolvimento de sistemas, seu grau de formalismo é muito baixo, o que pode resultar em sistemas não confiáveis e de má qualidade.

Atualmente não existe uma metodologia completa e sistemática que inclua formalismo no processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. A inclusão de métodos formais no desenvolvimento de sistemas dá mais rigor ao processo e evita ambigüidades. Os métodos formais podem ser aplicados em todo o processo de desenvolvimento de sistemas, apenas em algumas etapas, ou ainda, na especificação dos componentes mais críticos do sistema.

Outro ponto a ser destacado é o fato de que as ferramentas disponíveis no mercado não implementam todas as técnicas de mineração de dados, o que as tornam limitadas à solução de problemas específicos de mineração de dados.

As metodologias de implantação de Descoberta de Conhecimento são baseadas em iterações e interações de diferentes etapas (da análise do negócio à apresentação do conhecimento descoberto). Uma das principais características das mesmas está na participação direta dos especialistas no domínio e na ausência de técnicas de formalização que possam impedir a ambigüidade ou ineficácia de tais processos, considerando o não cumprimento de objetivos iniciais do sistema de Descoberta de Conhecimento.

No próximo capítulo é descrita a metodologia de pesquisa adotada neste trabalho.

3 METODOLOGIA DE DESENVOLVIMENTO DA PESQUISA

3.1 INTRODUÇÃO

Este capítulo tem como objetivo a apresentação da estrutura geral desta pesquisa, que é composta pelos seguintes elementos: o modelo da pesquisa e o processo de desenvolvimento da pesquisa.

3.2 MODELO DA PESQUISA

O modelo proposto integra as seguintes tecnologias: mineração de dados, modelagem orientada a objetos (UML), técnica de descrição formal (E-LOTOS) e agentes inteligentes. Essas tecnologias foram utilizadas como base para a definição de um modelo de formalização do processo de desenvolvimento de Sistemas de Descoberta de Conhecimento em Banco de dados (SDCBs), que tem como principal objetivo a obtenção de SDCBs confiáveis e de qualidade. A Figura 3.1 mostra este modelo.

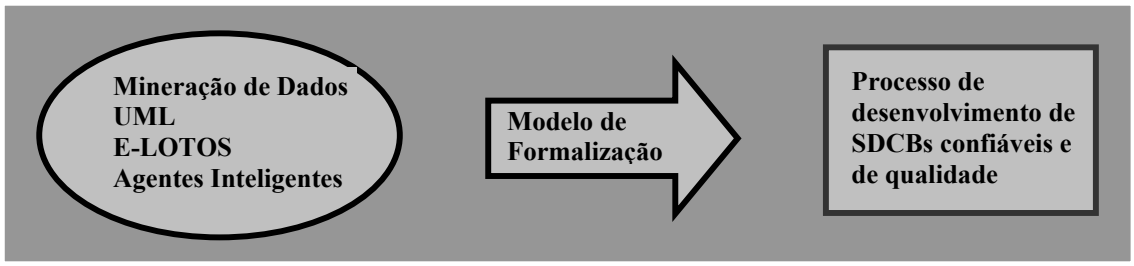


Figura 3.1: Modelo da Pesquisa

Esta pesquisa pode ser classificada como pesquisa aplicada, quantitativa e experimental. Uma pesquisa é considerada como aplicada porque "objetiva gerar conhecimentos para aplicação à solução de problemas específicos" (Silva e Menezes, 2000, p.20). Quantitativa porque o próprio objeto de estudo requer o uso de recursos e de técnicas estatísticas (percentagem, média, coeficiente de correlação etc.). Pesquisa experimental porque "determina-se um objeto de estudo, selecionam-se as variáveis capazes de influenciá-lo, definem-se as formas de controle e de observação dos efeitos que a variável produz no objeto" (Silva e Menezes, 2000, p. 21).

O objeto de estudo nesta pesquisa foi o processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. As variáveis capazes de influenciá-lo foram consideradas como sendo as tecnologias adotadas (mineração de dados, UML, E-LOTOS e agentes inteligentes) e as formas de controle e de observação dos efeitos que a variável produz no objeto foram analisadas na fase construtiva desta pesquisa.

3.3 PROCESSO DE DESENVOLVIMENTO DA PESQUISA

As principais etapas de pesquisa consideradas nesta tese foram: a escolha do tema, a revisão da literatura, a definição de uma metodologia de desenvolvimento de SDCBs, a especificação de um ambiente de implementação de SDCBs, a definição de um modelo de formalização do processo de desenvolvimento de SDCBs, a implementação de um protótipo do ambiente e a validação do modelo.

A Figura 3.2 representa as etapas do processo de desenvolvimento da pesquisa. Nessa figura, à esquerda são relacionadas as principais etapas da pesquisa e à direita os elementos envolvidos em cada etapa.

3.3.1 Escolha do Tema

O tema é fruto da identificação do problema inédito de pesquisa, justificável e fundamentado nas seguintes áreas: Engenharia de Software, Banco de Dados, Gestão da Informação e Inteligência Artificial.

Cada área tem relação com a proposta no âmbito de modelo, especificação, desenvolvimento e aplicação do objeto de pesquisa.

Foram determinadas as tecnologias específicas a serem aplicadas como base na obtenção de um processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados confiáveis e de qualidade, que são: mineração de dados, modelagem orientada a objetos (UML), métodos formais (E-LOTOS) e agentes inteligentes.

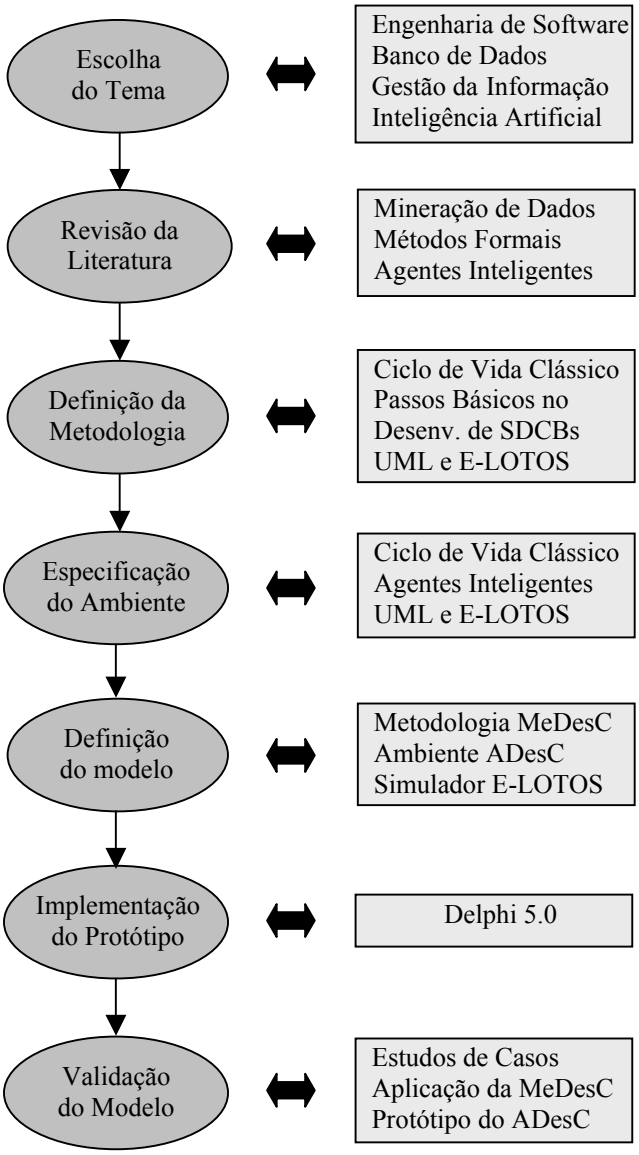


Figura 3.2: Processo de Desenvolvimento da Pesquisa

3.3.2 Revisão da Literatura

A revisão da literatura englobou conceitos e características das seguintes tecnologias adotadas na pesquisa: mineração de dados, métodos formais e agentes inteligentes.

Na revisão da literatura sobre mineração de dados, além dos conceitos e características, foram descritas tarefas e técnicas de mineração de dados mais comuns e relacionadas algumas propostas de metodologias de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados.

A linguagem de descrição formal utilizada nesta pesquisa foi E-LOTOS. Por isto, foi incluído o Anexo 9.1 que apresenta algumas características dessa linguagem.

A revisão da literatura e o conhecimento já sedimentado nas áreas de Engenharia de Software e de Banco de Dados deram o suporte necessário à definição da metodologia, do ambiente e do modelo de formalização do processo de desenvolvimento de SDCBs.

3.3.3 Definição da Metodologia

Nesta etapa, a metodologia de desenvolvimento de SDCBs foi definida detalhadamente, incluindo a descrição de como os modelos UML podem ser traduzidos para E-LOTOS.

As etapas definidas para a metodologia de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, aqui proposta, foram baseadas no ciclo de vida clássico, utilizado em outras metodologias de desenvolvimento de sistemas computacionais (Pressman, 1999) e nos passos básicos para o desenvolvimento desses sistemas (Groth, 1998), que incluem a aplicação de técnicas de mineração de dados.

A metodologia proposta nesta pesquisa utiliza modelos da linguagem de modelagem orientada a objetos UML na representação das características do sistema e a linguagem de especificação formal E-LOTOS para que a especificação do sistema possa ser verificada e validada.

3.3.4 Especificação do Ambiente

A especificação do ambiente de implementação de sistemas de descoberta de conhecimento em banco de dados foi realizada seguindo as etapas do ciclo de vida clássico de desenvolvimento de sistemas e usando modelos UML na representação das características do ambiente. A linguagem E-LOTOS foi usada na especificação formal do ambiente.

Além das tecnologias de modelagem orientada a objetos e de métodos formais, a especificação do ambiente tomou como base características de agentes inteligentes na definição dos objetos que compõem o ambiente.

3.3.5 Definição do Modelo

Nesta etapa definiu-se um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Esse modelo engloba a metodologia de desenvolvimento de SDCBs, denominada MeDesC; o ambiente de implementação de SDCBs, denominado ADesC e um simulador de especificações E-LOTOS.

3.3.6 Implementação do Protótipo

Um protótipo do ambiente foi implementado em linguagem Delphi 5.0, para tornar possível a validação do ambiente proposto através da realização de estudos de casos.

3.3.7 Validação do Modelo

A validação do modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados realizou-se através da definição e do desenvolvimento de estudo de caso. Esse estudo de caso tomou como base informações dos cursos de pós-graduação do Brasil contidas em Bancos de Dados da CAPES.

Após a definição do estudo de caso, o desenvolvimento do sistema de descoberta de conhecimento em banco de dados realizou-se de acordo com as etapas da metodologia proposta. Na fase de implementação do sistema utilizou-se o protótipo do ambiente.

4 MODELO GERAL PROPOSTO E METODOLOGIA DE DESENVOLVIMENTO DE SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

4.1 INTRODUÇÃO

A crescente exigência por sistemas confiáveis e de boa qualidade deu origem a muitas metodologias para desenvolvimento de sistemas computacionais. No entanto, essas metodologias, geralmente, não se aplicam adequadamente a alguns tipos de sistemas, tais como: sistemas de tempo real e sistemas de descoberta de conhecimento em banco de dados.

Apesar de existirem propostas de metodologias para o desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, essas metodologias não incluem formalismo, o que dificulta a obtenção de sistemas confiáveis e de qualidade.

Essas exigências podem ser satisfeitas em sistemas computacionais complexos, com o uso de Modelagem Orientada a Objetos (MOO) e de uma Técnica de Descrição Formal (TDF). A MOO facilita a representação de objetos do mundo real em objetos computacionais. As TDFs possibilitam a verificação e a validação da especificação do sistema.

Atualmente, existem ferramentas de mineração de dados disponíveis no mercado que solucionam determinados tipos de problemas de descoberta de conhecimento em banco de dados, conforme pode ser visto na Tabela 2.3 do Capítulo 2. No entanto, essas ferramentas, além de apresentarem limitações, são baseadas em métodos '*ad-hoc*', o que pode levar a resultados ambíguos.

Para suprir as necessidades relacionadas acima, este capítulo apresenta um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, que engloba uma metodologia de desenvolvimento de sistemas de descoberta de conhecimento, denominada MeDesC; um ambiente de implementação dos modelos resultantes da aplicação dessa metodologia, denominado ADesC e uma ferramenta de simulação e validação de especificações de sistemas em E-LOTOS. Também, neste capítulo, são descritas, detalhadamente, todas as etapas da metodologia MeDesC.

4.2 O MODELO GERAL PROPOSTO

Como foi visto anteriormente, o principal objetivo desta tese é definir um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Este modelo é representado na Figura 4.1 abaixo.

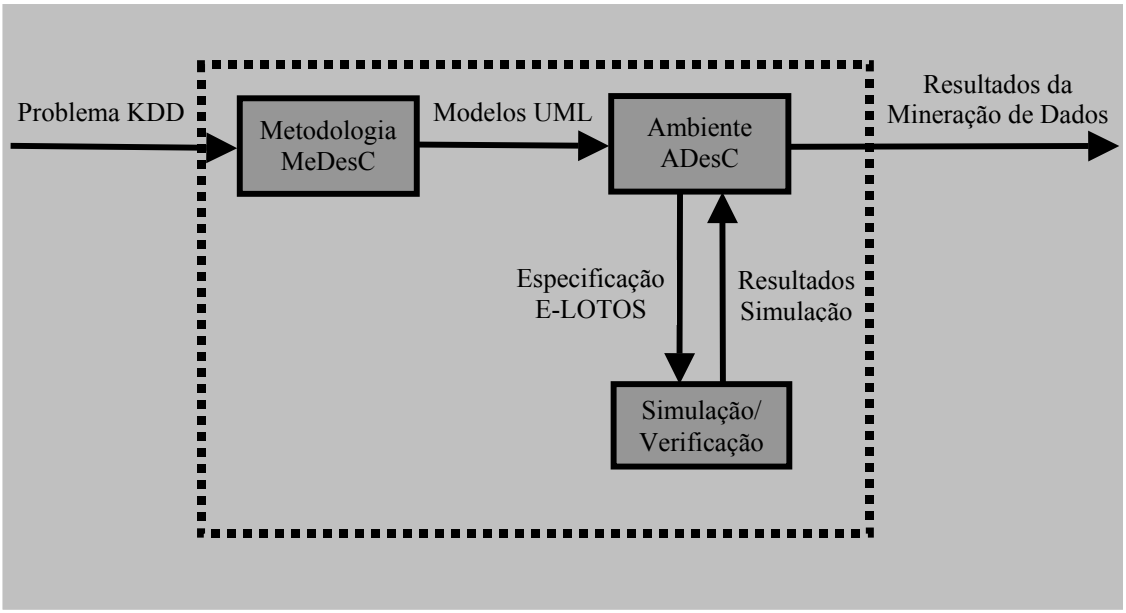


Figura 4.1: Modelo Geral Proposto

Ao observar a Figura 4.1, pode-se entender que o sistema de descoberta de conhecimento é desenvolvido partindo-se de um problema específico de KDD (*Knowledge Discovery in Databases*). Sobre esse problema é aplicada a metodologia MeDesC, que tem como resultado modelos UML representando as características específicas do sistema em desenvolvimento. Esses modelos UML resultantes são usados como argumentos de entrada no ambiente ADesC que os traduz na linguagem E-LOTOS. A especificação formal do ambiente pode ser validada através da utilização de uma ferramenta de simulação e verificação de especificações E-LOTOS disponível.

A metodologia MeDesC integra UML e E-LOTOS. UML (*Unified Modeling Language*) é uma linguagem de modelagem unificada baseada em modelos de metodologias orientadas a objetos (Booch et al, 1999). E-LOTOS (*Enhancements to Language Of Temporal Ordering Specification*) é uma TDF que permite a especificação formal e modular de sistemas e a representação de tempo; foi padronizada pela ISO (*International Standards Organization*) (ISO/IEC, 1997). Conforme foi visto na seção

2.3.3 do Capítulo 2, métodos formais podem ser aplicados em níveis variados de rigor; assim, na metodologia MeDesC, o nível de rigor aplicado foi 1, considerando que foi incluída uma etapa de formalização no processo de desenvolvimento de sistemas computacionais. As etapas dessa metodologia são descritas nas próximas seções.

O analista de sistemas poderá utilizar a metodologia MeDesC por completo e implementar o seu sistema de descoberta de conhecimento em banco de dados em qualquer linguagem de programação que lhe for mais conveniente, ou então, poderá utilizar a metodologia MeDesC até a fase de projeto informal e implementar seu sistema no ambiente ADesC.

O ambiente ADesC (descrito mais detalhadamente no Capítulo 5) tem como objetivos principais: auxiliar o analista de sistemas na etapa de implementação de sistemas de descoberta de conhecimento em banco de dados, oferecendo-lhe facilidades na construção desses sistemas; traduzir modelos UML para E-LOTOS e oferecer ao usuário final um ambiente interativo e amigável que gera informações relevantes à tomada de decisão da empresa. Essas informações são obtidas pelo ambiente através da aplicação de técnicas de mineração de dados.

O desenvolvimento do ambiente ADesC é baseado no ciclo de vida clássico de sistemas. No entanto, foi incluída a etapa de projeto formal para que a especificação do ambiente possa ser simulada e verificada, resultando, assim, em um ambiente confiável e de qualidade.

Na especificação do ambiente ADesC, também são usados modelos UML nas etapas de análise do sistema e de projeto informal. Os modelos resultantes da fase de projeto informal são traduzidos para E-LOTOS (Anexo 9.4).

O ambiente ADesC baseia-se na tecnologia de agentes para facilitar a busca e a seleção de informações em grandes bancos de dados, que podem estar espalhados por diferentes nós de redes de computadores; a escolha da técnica de mineração de dados mais adequada e a análise dos resultados. Os objetos definidos no ambiente possuem características que os identificam como pertencentes às classes de agentes inteligentes, agentes móveis ou agentes de software. No entanto, a tecnologia de agentes foi aplicada apenas na especificação do ambiente ADesC, o protótipo e o estudo de casos realizado não se basearam nesta tecnologia devido à complexidade de sua implementação.

Todas as fases básicas de um sistema de descoberta de conhecimento em banco de dados são realizadas no ambiente ADesC. São elas: pré-processamento, construção de um modelo e pós-processamento. Em todas essas fases, o ambiente interage com o

usuário para que este possa selecionar os atributos necessários, definir os tipos de transformações sobre os dados, escolher a técnica de mineração de dados a ser aplicada e realizar a análise dos resultados obtidos.

Na fase de pré-processamento, o ambiente ADesC possui uma função não encontrada em outras ferramentas de mineração de dados, que é a função de Tradução para E-LOTOS. Esta função traduz os modelos UML, que representam as características do sistema projetado pelo analista de sistemas, para a linguagem E-LOTOS; possibilitando, assim, a formalização da especificação do sistema.

Assim como é descrito na literatura (Berry e Linoff, 1997), (Groth, 1998), (Freitas, 1998), (Feldens et al, 1998) etc, a fase de pré-processamento do ambiente ADesC engloba as tarefas de preparação e de transformação de dados. Essas tarefas são realizadas com a ajuda de agentes móveis e de agentes de software. O conjunto de dados preparado e transformado é armazenado em uma estrutura de dados denominada *Dataset*, que é definida de acordo com o sistema desenvolvido. A função de definição das formas de povoamento dos dados, realizada pelo ambiente ADesC, também pode ser considerada como pertencente à fase de pré-processamento. Essa função é importante na manutenção de dados atualizados no *DataSet*. As informações sobre os dados selecionados, sobre as transformações realizadas nos dados e sobre as definições das formas de povoamento dos dados, são armazenadas em tabelas do metadados.

Na fase de construção de um modelo é aplicada uma técnica de mineração de dados sobre o *Dataset*, adequada ao tipo de problema KDD em estudo. A escolha dessa técnica é realizada pelo usuário (analista de sistemas) com a ajuda de um agente inteligente.

Após o modelo ser construído, através da aplicação de uma técnica de mineração de dados, os resultados são armazenados para que, posteriormente, o usuário final possa analisá-los na fase de pós-processamento. A análise dos resultados também é realizada com a ajuda de um agente inteligente, que poderá, por exemplo, eliminar resultados menos relevantes.

4.3 A METODOLOGIA MeDesC

As principais etapas da metodologia MeDesC são baseadas no ciclo de vida clássico de desenvolvimento de sistemas (Pressman, 1999) e nos passos básicos para o desenvolvimento de sistemas de descoberta de conhecimento em banco de dados

(Groth, 1998). São elas: análise do sistema, projeto informal, projeto formal, implementação e análise dos resultados. A Figura 4.2 representa estas etapas.

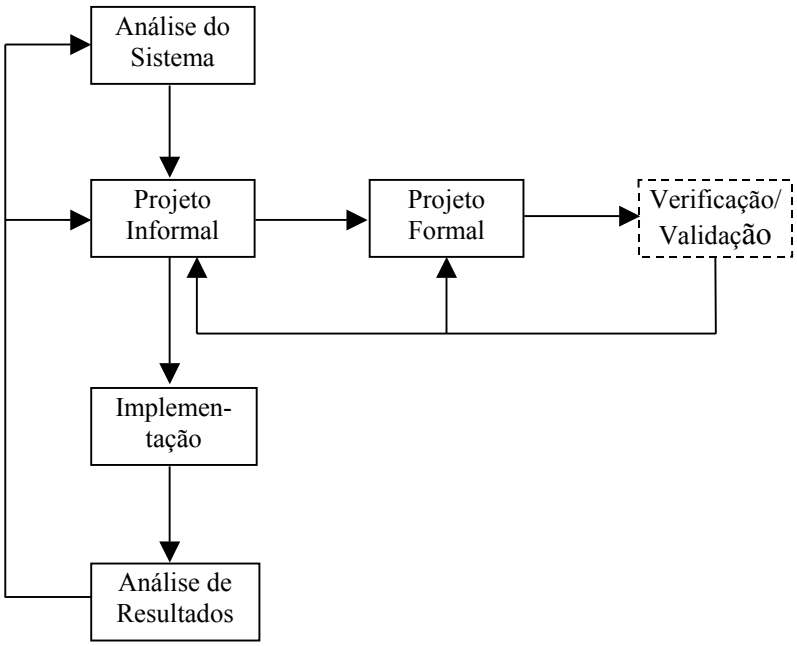


Figura 4.2: Etapas da Metodologia MeDesC

Apesar das etapas de análise do sistema, projeto informal e implementação constarem na grande maioria das metodologias de desenvolvimento de sistemas computacionais, os objetivos dessas etapas na metodologia MeDesC são diferentes daqueles definidos para outras metodologias.

A metodologia proposta define um processo iterativo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, por prever a necessidade de retornar às etapas de projeto informal e projeto formal caso seja encontrado algum erro durante a verificação/validação do sistema. Após a análise dos resultados, pode-se retornar às etapas de análise do sistema e de projeto informal para que seja definido um novo sistema ou ajustado o sistema já desenvolvido. Neste caso, os resultados obtidos podem ser utilizados como base.

Na etapa de análise do sistema e de projeto informal são utilizados diagramas UML (*Unified Modeling Language*) (Booch et al, 1999) para representar os objetos do sistema, seus comportamentos e suas interações.

A UML foi escolhida para ser usada na metodologia como forma de representação das características do sistema por ser uma linguagem de modelagem orientada a objetos padrão, definir diferentes tipos de diagramas que facilitam o

desenvolvimento de sistemas e, também, por separar claramente os aspectos estáticos dos dinâmicos e funcionais e permitir fáceis mudanças, além de ser uma linguagem unificada para modelagem de objetos.

Como foi visto no Capítulo 2, técnicas de descrição formal podem ser utilizadas na especificação de sistemas em diferentes níveis de rigor. Elas podem ser incluídas em todas ou apenas em alguma(s) das etapas do processo de desenvolvimento de sistemas. No entanto, é mais interessante inseri-las nas primeiras etapas. Como já foi dito anteriormente, na metodologia MeDesC foi aplicado o nível 1 de utilização de método formal.

A etapa de projeto formal foi incluída na metodologia para tornar mais rigoroso o processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. A formalização da especificação do sistema é realizada através do mapeamento dos modelos UML, construídos na fase de projeto informal, para a linguagem de especificação formal E-LOTOS. A especificação formal do sistema poderá ser verificada e validada com a utilização de uma ferramenta adequada[1] e, no caso de ser detectado algum erro, deve-se retornar à etapa de projeto informal ou à etapa de projeto formal.

A implementação é realizada tomando como base os diagramas UML, construídos na etapa de projeto informal, e utilizando o ambiente ADesC, definido no Capítulo 5, ou algum ambiente de programação que melhor convir ao analista de sistemas; mas só após a especificação formal ser validada, para que seja garantida a obtenção de um sistema de descoberta de conhecimento confiável e de boa qualidade.

Nas próximas seções são descritas, detalhadamente, as estratégias a serem adotadas em cada etapa da metodologia, de acordo com as técnicas utilizadas.

4.4 ANÁLISE DO SISTEMA

A primeira etapa da metodologia, a análise do sistema, tem como principais objetivos: definir tipos de investigações a serem realizadas com a aplicação de técnicas de mineração de dados e identificar as fontes de dados necessárias nessas investigações.

[1] Até o presente momento, não existe uma ferramenta de verificação e validação de especificação formal de sistemas em E-LOTOS.

As principais atividades desta etapa, representadas na Figura 4.3, são:

1) Descrição Inicial do problema;
2) Definição dos objetivos do sistema;
3) Construção de diagramas de classes do(s) sistema(s) em operação;
4) Construção do diagrama de classes corporativo;

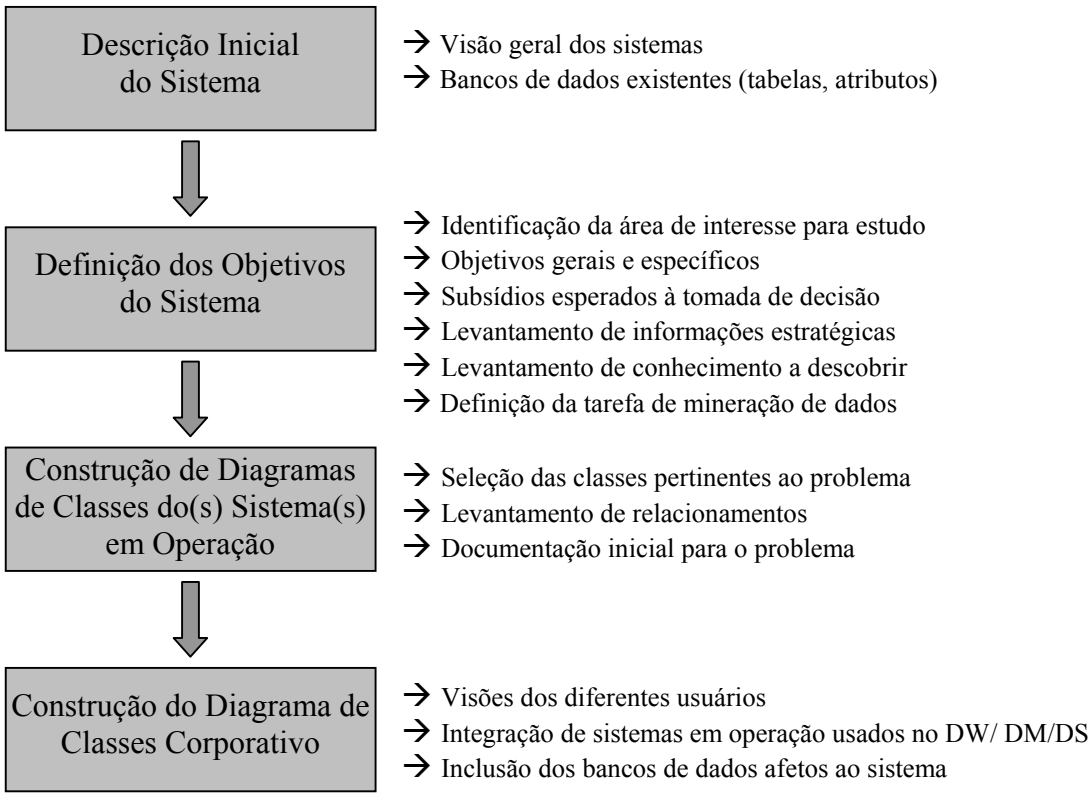


Figura 4.3: Atividades da Etapa Análise do Sistema

A documentação gerada nesta etapa é o Documento de Análise contendo uma descrição do problema, os principais objetivos a serem atingidos com o sistema a ser desenvolvido, os diagramas de classes individuais e o diagrama de classes corporativo.

4.4.1 Descrição Inicial do Problema

A etapa de análise do sistema inicia-se com a descrição do problema a ser solucionado. Na verdade, pode não existir um problema real a ser solucionado, considerando que a mineração de dados pode ser aplicada como um processo de descoberta, no qual nem sempre é feito algum tipo de suposição antecipada.

Na descrição do problema deve-se dar uma visão geral do funcionamento do(s) sistema(s) em operação, pertencentes ao domínio do problema, e dos bancos de dados existentes como, por exemplo, relacionar as tabelas e atributos de bancos de dados operacionais e a periodicidade de sua atualização.

4.4.2 Definição dos Objetivos do Sistema

Após ser feita a descrição do problema, deve-se identificar a área de interesse para estudo, definir os objetivos gerais e específicos que deverão ser alcançados com o desenvolvimento do sistema e determinar como os resultados obtidos podem ser usados no suporte à tomada de decisão na empresa.

Na definição dos objetivos deve–se ter uma idéia sobre que tipos de informações seriam estrategicamente interessantes de serem obtidas e sobre como o conhecimento descoberto poderá ser aplicado na empresa.

Com a definição dos objetivos gerais e específicos a serem alcançados e tendo conhecimento dos tipos mais interessantes de informações estratégicas a serem descobertos, é possível determinar a tarefa de mineração de dados a ser resolvida.

4.4.3 Construção de Diagramas de Classes do(s) Sistema(s) em Operação

Um diagrama de classes mostra a estrutura estática de dados do sistema em operação existente (Booch et al, 1999). Este tipo de diagrama descreve as classes de objetos do sistema e os vários tipos de relacionamentos existentes entre os objetos.

Na etapa de análise do sistema devem ser construídos diagramas de classes segundo as visões de diferentes usuários e/ou de diferentes sistemas em operação utilizados como base para o sistema em desenvolvimento. Nesta etapa, a classe de objetos, representada por um retângulo, pode conter somente o seu nome; sendo os seus atributos e suas operações incluídos em etapas posteriores.

As informações para os diagramas de classes provêm da definição inicial do problema, do perfeito domínio da aplicação e do conhecimento geral do mundo real. Se o(s) sistema(s) em operação estiver(em) bem documentado(s), as documentações existentes podem ser utilizadas na etapa de análise do sistema.

Antes de serem construídos os diagramas de classes, é necessário definir quais classes de objetos pertencem e quais não pertencem ao escopo do modelo de dados do

novo sistema. Isto pode ser feito com base na descrição inicial do problema e nos objetivos do sistema, obtidos anteriormente.

4.4.4 Construção do Diagrama de Classes Corporativo

O Diagrama de Classes Corporativo é construído tomando como base os diversos diagramas de classes individuais, construídos anteriormente. Ele representa a integração dos sistemas em operação utilizados na construção do *data warehouse* (DW), *data mart* (DM) ou *data set* (DS). A Figura 4.4 representa esse diagrama.

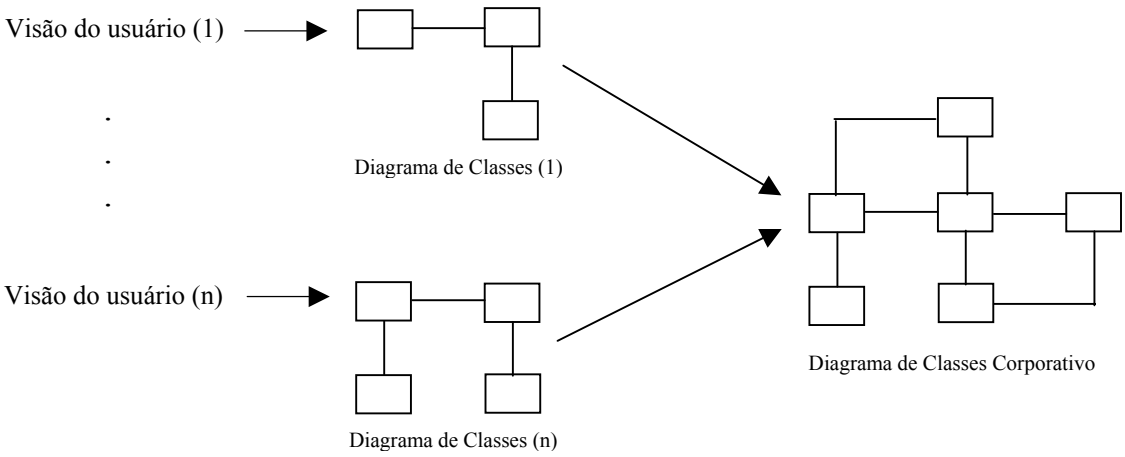


Figura 4.4: Diagrama de Classes Corporativo

Na construção desse diagrama, deve-se considerar apenas as classes de objetos que representam os bancos de dados contendo os dados necessários ao sistema de descoberta de conhecimento em desenvolvimento.

4.5 PROJETO INFORMAL

A etapa de projeto informal consiste na seleção dos atributos; transformação dos dados, projeto do metadados; projeto do DW/DM/DS; definição de técnica estatística; escolha de uma ou mais técnicas de mineração de dados; na descrição do

comportamento dos objetos através de diagramas UML e determinação da forma de povoamento[2] do DW/DM/DS.

As principais atividades desta etapa são:

1) Seleção dos atributos;
2) Definição das transformações dos dados;
3) Projeto de uma estrutura de metadados;
4) Projeto do DW/DM/DS;
5) Definição de técnica de amostragem;
6) Escolha de técnica de mineração de dados;
7) Construção dos diagramas de classes;
8) Construção dos diagramas de estados;
9) Construção dos diagramas de colaboração;
10) Determinação da forma de povoamento do DW/DM/DS;

A Figura 4.5 representa as atividades desta etapa.

Nesta etapa será gerado o Documento de Projeto Informal, contendo o projeto e a definição da forma de povoamento do DW/DM/DS, a definição de técnica estatística, a(s) técnica(s) mineração de dados a ser(em) aplicada(s) e os diagramas de classes, de estados e de colaboração.

4.5.1 Seleção dos Atributos

A seleção dos atributos deve ser feita tomando como base as características das classes de objetos definidas no Diagrama de Classes Corporativo e/ou os itens de dados existentes nos bancos de dados operacionais e, também, a tarefa de mineração de dados a ser solucionada. Os atributos a serem selecionados deverão ser aqueles estritamente necessários de serem mantidos no DW/DM/DS para o cumprimento dos objetivos do sistema, definidos anteriormente.

Os atributos selecionados serão representados no Diagrama de Classes Corporativo, juntamente com as classes de objetos a que eles pertencem.

[2] O termo povoamento é utilizado como sinônimo de atualização e envolve a definição de como e quando as atualizações realizadas nos bancos de dados operacionais devem ser refletidas no DW.

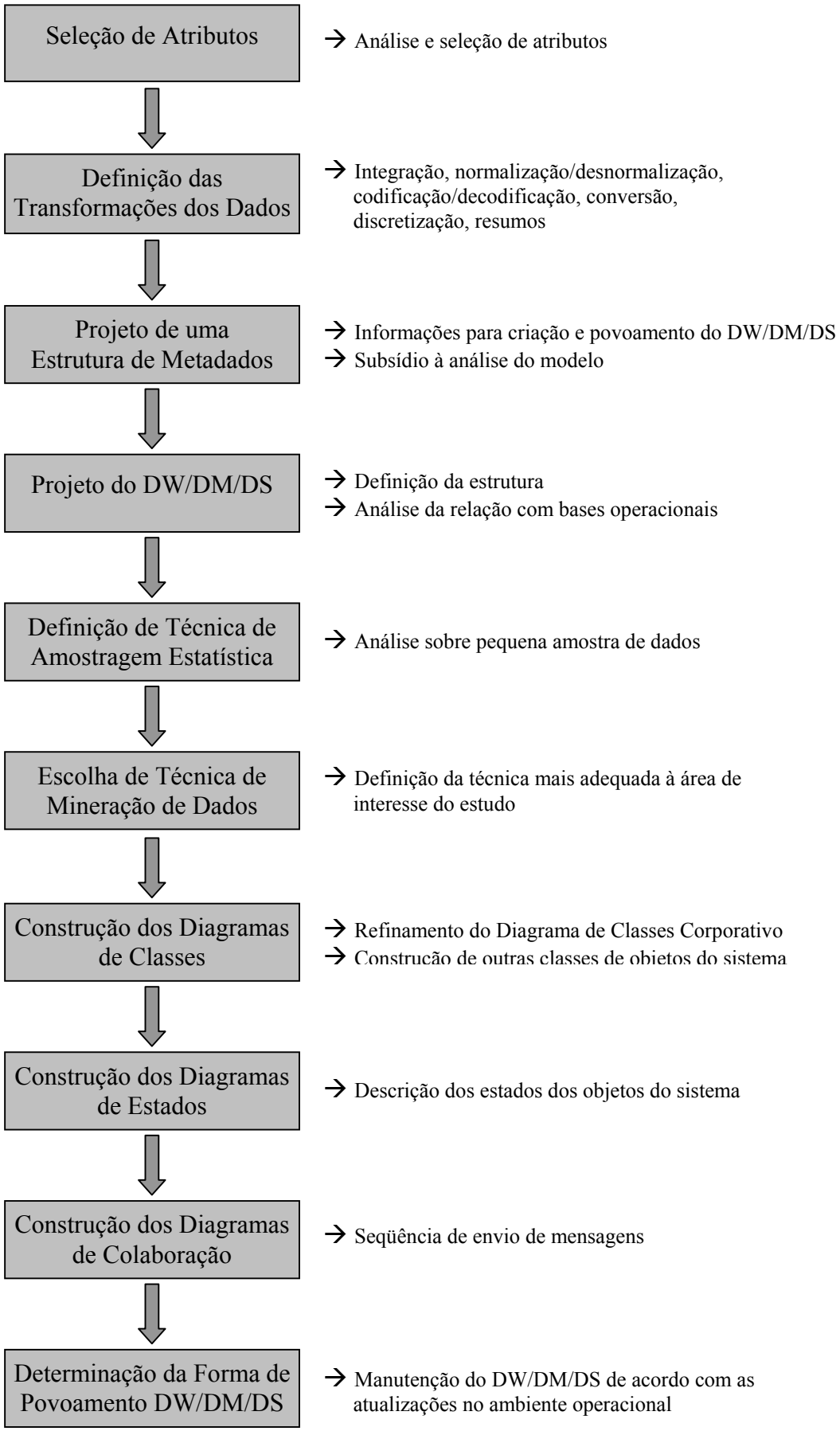


Figura 4.5: Atividades da Etapa Projeto Informal

Freitas (1998) descreve duas abordagens básicas para a seleção de atributos. São elas: *wrapper* e filtro.

A abordagem *wrapper* usa um algoritmo de mineração de dados para avaliar a qualidade de um conjunto de atributos. A função de avaliação é uma medida de desempenho do algoritmo (tipicamente, a taxa de acerto obtida pelo algoritmo), usando os atributos selecionados. Assim sendo, a seleção de atributos é otimizada para aquele algoritmo.

A abordagem filtro é independente de qualquer algoritmo de mineração de dados. Um método bem simples para a abordagem filtro é analisar a distribuição das classes para diferentes valores de cada atributo de previsão, montando uma tabela bidimensional. Nessa tabela, as linhas indicam valores do atributo de previsão, as colunas indicam valores (ou classes) do atributo meta e cada célula contém o número de tuplas satisfazendo os respectivos valores dos atributos de previsão e meta. Exemplo:

sexo	compra? sim	não
M	20	30
F	20	30

salário	compra? sim	não
alto	18	2
médio	20	10
baixo	2	48

Intuitivamente, salário é melhor atributo de previsão do que sexo, pois, sabendo-se o valor de salário fica mais fácil prever o valor do atributo meta “compra”, em comparação com saber o valor de sexo. Logo o atributo salário seria selecionado, enquanto o atributo sexo não.

4.5.2 Definição das Transformações dos Dados

De acordo com Inmon (1997), à primeira vista, quando os dados são movidos do ambiente herdado para o ambiente de DW, parece que nada além de simples extrações de dados de um local para o outro está ocorrendo. Contudo, as primeiras impressões podem ser muito enganosas. O que em um primeiro momento parece ser nada mais do que a movimentação de dados de um local para outro, transforma-se, rapidamente, em uma grande e complexa tarefa.

Os dados existentes nos bancos de dados operacionais deverão sofrer transformações, durante a construção do DW/DM/DS, para se adequarem às necessidades do sistema em desenvolvimento. Os tipos comuns de transformações são:

- Integração de dados;
- Normalização/desnormalização;
- Codificação/decodificação;
- Conversões;
- Discretização;
- Inclusão de novos campos de dados;
- Definição de resumos.

A integração de dados muitas vezes é necessária porque geralmente existem várias fontes de dados a serem utilizadas na construção do DW.

O resultado do processo de construção do modelo de dados consiste em uma série de tabelas, cada uma das quais contendo chaves e atributos. A existência de muitas tabelas quase sempre é necessária na normalização dos bancos de dados, mas pode prejudicar o desempenho e o custo do sistema devido aos recursos de entrada/saída (E/S) consumidos, tanto em termos de acesso aos dados, como em termos de acesso ao índice para a localização dos dados. Muitas vezes é necessário desnormalizar o DW para diminuir o número de E/S. Segundo Inmon (1997), existem algumas técnicas de projeto que podem poupar E/S. São elas:

- Intercalação de tabelas;
- Criação de uma cadeia de dados;
- Introdução intencional de dados redundantes;
- Separação de dados que apresentem probabilidades de acesso muito diferentes;
- Introdução de dados derivados (calculados);
- Índice "criativo" (o índice criativo produz um perfil de itens de interesse do usuário final).

Em alguns casos, a conversão do formato dos dados precisa ser feita, de EBCDIC para ASCII ou vice-versa. Também, pode ser necessário reformatar os dados. Como um exemplo simples, dados de entrada sobre data são lidos como AA/MM/DD e são gravados no arquivo de saída como DD/MM/AA. Freqüentemente, a re-formatação

de dados operacionais antes que eles estejam prontos para passar para o DW é muito mais complexa do que esse exemplo simples.

A utilização de dados contínuos é, muitas vezes, impossibilitada por características inerentes a determinadas técnicas de mineração de dados. Assim, pode haver necessidade de tornar dados contínuos em dados discretos.

Freqüentemente é necessário resumir dados. Vários registros operacionais de entrada são combinados em um único registro do DW. Para efetuar o resumo, os registros de entrada detalhados, candidatos ao resumo, devem estar dispostos em uma seqüência adequada.

As transformações a serem realizadas são representadas nos Diagramas de Estados.

4.5.3 Projeto de uma Estrutura de Metadados

A estrutura de metadados deve fornecer informações a serem usadas na criação e povoamento do DW/DM/DS, bem como informações que serão usadas pelos analistas no entendimento dos dados e na construção dos modelos.

Sem a existência de uma estrutura de metadados, as tarefas de seleção de atributos e de definição das transformações dos dados terão que ser realizadas toda vez que houver a necessidade de povoamento do DW/DM/DS e sempre antes de ser feita uma investigação através da aplicação de uma técnica de mineração de dados.

4.5.4 Projeto do DW/DM/DS

Deve ser definida a estrutura do DW/DM/DS necessária para o armazenamento dos dados utilizados na aplicação de técnicas de mineração de dados. Caso já exista o DW ou um *data mart* e ainda seja preciso resumir mais os dados, então será necessário definir apenas tabelas para armazenar o conjunto de dados (*data set*) selecionados anteriormente.

Uma estrutura de *data set* pode ser definida, também, independentemente da existência de um DW/DM; ou seja, o *data set* pode ser gerado a partir de bancos de dados operacionais. A opção da utilização de um *data set* no lugar de um DW/DM pode estar associada à facilidade de sua modelagem e sua construção e, também, à especificidade da investigação a ser realizada.

4.5.5 Definição de Técnica de Amostragem Estatística

Uma técnica de amostragem estatística pode ser aplicada sobre as bases de dados para se obter uma pequena amostra dos dados.

Uma pequena amostra de dados, às vezes, é necessária para evitar a aplicação de uma técnica de mineração de dados sobre um volume muito grande de dados, o que causaria um custo muito alto e/ou muitas horas de processamento. Um modelo pode ser criado de uma pequena amostra e então ser validado sobre o conjunto completo de dados. Se um conjunto de dados de uma amostra é válido, então ele é capaz de prever resultados para outras amostras de dados.

4.5.6 Escolha de Técnica de Mineração de Dados

A escolha de técnica de mineração de dados deve tomar como base a área de interesse para estudo, os objetivos do sistema a ser implementado, os tipos de dados a serem analisados, as características das técnicas disponíveis e a tarefa de mineração de dados a ser resolvida.

O Capítulo 2 descreveu as técnicas de mineração de dados normalmente usadas na busca de conhecimento em banco de dados.

4.5.7 Construção dos Diagramas de Classes

Um diagrama de classes corporativo é construído na etapa de análise do sistema para representar os bancos de dados do(s) sistema(s) atual(is) em operação que foram tomados como base para o desenvolvimento do sistema de descoberta de conhecimento em banco de dados. Após esse diagrama ser refinado, ele é construído novamente para ilustrar as classes de objetos que representam as tabelas desses bancos de dados, seus relacionamentos, os atributos selecionados e os métodos definidos. Os diagramas de classes do metadados e do DW/DM/DS também devem ser construídos nesta fase.

4.5.8 Construção dos Diagramas de Estados

Os diagramas de estados descrevem todos os possíveis estados de um objeto particular e como as mudanças de estados ocorrem como resultado de eventos que alcançam o objeto (Booch et al, 1999).

Um diagrama de estados deve ser construído para cada classe de objetos definida anteriormente, para um melhor entendimento do sistema e para facilitar o desenvolvimento das fases seguintes.

4.5.9 Construção dos Diagramas de Colaboração

Os diagramas de colaboração mostram a seqüência na qual as mensagens são enviadas entre os objetos do sistema (Booch et al, 1999).

Um diagrama de colaboração pode ser construído para cada método das classes de objetos especificadas para o sistema em desenvolvimento. A necessidade ou não da construção de diagramas de colaboração dependerá da complexidade do sistema, principalmente das dificuldades que poderão existir na transformação dos dados.

4.5.10 Determinação da Forma de Povoamento do DW/DM/DS

A criação do DW/DM/DS é feita, geralmente, com dados existentes nos bancos de dados dos sistemas em operação. Uma vez criado, o DW/DM/DS, deverá ser atualizado quando houver alguma atualização no ambiente operacional.

O tempo necessário para que uma alteração sobre dados do ambiente operacional reflita no DW é referido como "ciclicidade de dados". Segundo Inmon (1997), como regra, pelo menos 24 horas deveriam se passar entre o momento em que a alteração é observada pelo ambiente operacional e sua repercussão no DW. Há várias razões para a necessidade de colocar uma "dobra de tempo"[3] nos dados.

A primeira razão é que uma dobra de tempo de 24 horas pode facilmente ser alcançada com tecnologia convencional. No entanto, uma razão mais forte é que a dobra de tempo impõe uma certa disciplina aos ambientes, não havendo a tentação de efetuar processamento operacional no DW e processamento do DW no ambiente operacional.

[3] Dobra de tempo: é o intervalo mínimo de tempo de espera para que uma alteração realizada no ambiente operacional possa ser refletida no *data warehouse*.

Um outro benefício da dobra de tempo é a oportunidade para os dados estabilizarem-se antes de serem movidos para o DW.

De acordo com Inmon (1997), existe uma interação básica de negócios que faz com que o DW se torne povoado por dados. Essa interação básica é chamada de interação EVENTO ⟶ INSTANTÂNEO. Neste tipo de interação, um evento dispara um instantâneo ("*snapshot*") de dados (normalmente no ambiente operacional), que por sua vez é movido para o ambiente do DW.

Um instantâneo é uma estrutura de dados que apresenta quatro componentes básicos: uma chave, uma unidade de tempo, dados primários que se relacionam somente com a chave e dados secundários que são capturados como parte do processo de criação do instantâneo e não possuem relacionamento direto com os dados primários ou com a chave.

O evento de negócios que dispara um instantâneo pode ser a ocorrência de alguma atividade importante como a realização de uma venda, a entrada de um item em estoque, uma chamada telefônica ou a entrega de uma carga. Esse tipo de evento de negócios é chamado de *evento gerado pela atividade*. O outro tipo de evento de negócios que pode disparar um instantâneo no DW é a marcação da passagem normal do tempo, como o final do dia, o final da semana ou o final do mês. Esse tipo de evento de negócios é chamado de *evento gerado pelo tempo*.

4.6 PROJETO FORMAL

O objetivo da etapa de projeto formal é dar maior rigor ao processo de desenvolvimento de sistema de descoberta de conhecimento em banco de dados. A utilização de uma técnica de descrição formal torna possível a realização de verificação e validação do sistema especificado. Nesta etapa, é adotada a linguagem de especificação formal E-LOTOS (Anexo 9.1).

E-LOTOS é uma linguagem de especificação formal adequada por permitir a especificação de sistemas de forma modular, por possibilitar a representação de tempo que pode ser utilizada para mostrar, por exemplo, quando o DW/DM/DS é povoado pelos dados atuais e, principalmente, por permitir a verificação e a validação do sistema.

No projeto formal deve ser construído um modelo formal em E-LOTOS, manual ou automaticamente, a partir dos diagramas de UML e, após o modelo formal estar completo, ele poderá ser verificado e validado através de uma ferramenta de

simulação/validação a ser desenvolvida. A seguir são descritos os passos para a construção desse modelo e as estratégias a serem utilizadas.

4.6.1 Mapeamento dos Diagramas UML para E-LOTOS

A formalização de modelos orientados a objetos deve ser realizada através da tradução das características desses modelos para as representações existentes na linguagem formal utilizada.

No entanto, existem dificuldades na incorporação de TDFs (Técnicas de Descrição Formal) no desenvolvimento orientado a objetos usando uma linguagem de modelagem. Por exemplo, o comportamento dinâmico global de sistemas baseados em objetos é difícil de ser especificado com as TDFs atuais. Isto ocorre porque a maioria dessas TDFs não incorpora características desses sistemas, tais como modularidade, herança, encapsulamento etc.

Essas dificuldades podem ser superadas através da definição de uma forma de mapeamento entre os conceitos da modelagem orientada a objetos e da TDF utilizada.

Com a utilização de UML e de E-LOTOS, o resultado deste mapeamento é um modelo formal que contém uma especificação do sistema em E-LOTOS, a partir de características apresentadas nos diagramas UML. O modelo formal integra, assim, as propriedades estáticas e dinâmicas do sistema. Como a especificação formal obtida é executável, pode-se usar a prototipagem para validá-la.

A construção do modelo formal em E-LOTOS é realizada através dos seguintes passos:

1) Divisão do sistema em módulos;
2) Definição do modelo de comunicação;
3) Declaração dos módulos.

A Figura 4.6 representa esses passos.

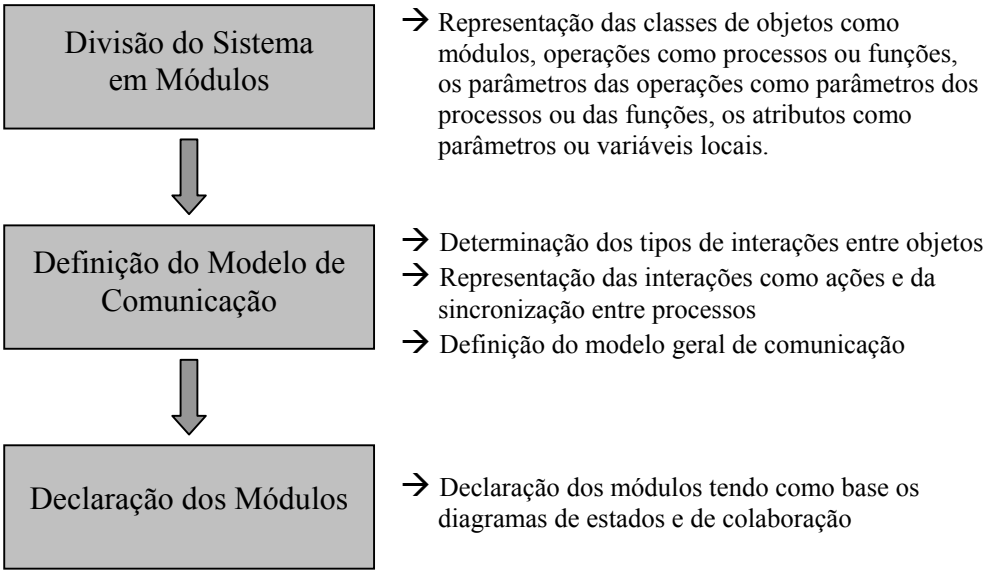


Figura 4.6: Atividades da Etapa Projeto Formal

4.6.2 Divisão do Sistema em Módulos

A linguagem E-LOTOS permite a divisão do sistema em módulos, o que facilita a especificação de um sistema tendo como base os diagramas UML. Neste primeiro passo é utilizado o diagrama de classes.

As classes de objetos são declaradas como módulos, suas operações como processos ou funções, os parâmetros das operações são os parâmetros dos processos ou das funções e os atributos das classes são declarados como parâmetros ou como variáveis locais dos processos e das funções. O problema dessas opções de declarar atributos é que eles sempre terão que ser declarados quando forem utilizados em um processo ou em uma função. Não é possível declarar variáveis globais em módulos. A linguagem E-LOTOS permite apenas a declaração de constantes em nível de módulos.

O diagrama de classes permite a representação de relacionamentos entre objetos através de ligações e associações. A herança é um tipo de associação poderosa por permitir o compartilhamento de características entre classes de objetos, tornando possível a reutilização de código.

Na linguagem E-LOTOS a reutilização de código é possível com a declaração de superclasses de objetos como módulos e importando suas características em outros módulos (subclasses de objetos) através da cláusula "*import*". Desta forma, é possível

representar herança simples e herança múltipla. No entanto, apenas os métodos (processos e funções), tipos de dados e as constantes da(s) superclasse(s) serão herdados diretamente pela subclasse, visto que os atributos só podem ser declarados como parâmetros ou variáveis locais de processos e funções.

4.6.3 Definição do Modelo de Comunicação

Para que o comportamento de um sistema possa ser especificado, é preciso que sejam determinados os tipos de interações entre objetos. Essas interações são representadas nos diagramas de interação de UML como mensagens e na linguagem E-LOTOS como ações.

A comunicação/interação entre objetos nos diagramas de interação de UML é assíncrona. O conceito de comunicação assíncrona relaciona-se ao fato de uma mensagem ser uma transmissão ou informação unidirecional de um objeto para outro. Um objeto que envia uma mensagem a outro objeto pode esperar uma resposta, sendo essa resposta representada como um retorno enviado pelo segundo objeto, que pode resolver enviá-lo ou não. Por outro lado, o modelo E-LOTOS baseia-se no sincronismo, ou seja, para que haja uma comunicação entre processos é necessário que os processos comunicantes se sincronizem em uma determinada porta de comunicação.

A especificação de um sistema na linguagem E-LOTOS a partir dos diagramas UML requer uma modificação do modo de comunicação de assíncrono para síncrono. Desta forma, o envio e a resposta de uma mensagem serão feitos através da sincronização entre o objeto emissor e o objeto receptor. Isto causa a diminuição do paralelismo.

A definição do modelo de comunicação do sistema a ser desenvolvido deve tomar como base o modelo geral de comunicação apresentado na Figura 4.7.

No modelo geral de comunicação, os usuários interagem com o ambiente de descoberta de conhecimento para definir o tipo de sistema que necessita. Essa interação é realizada através das portas de comunicação de entrada e de saída denominadas *pent* e *psai*, respectivamente.

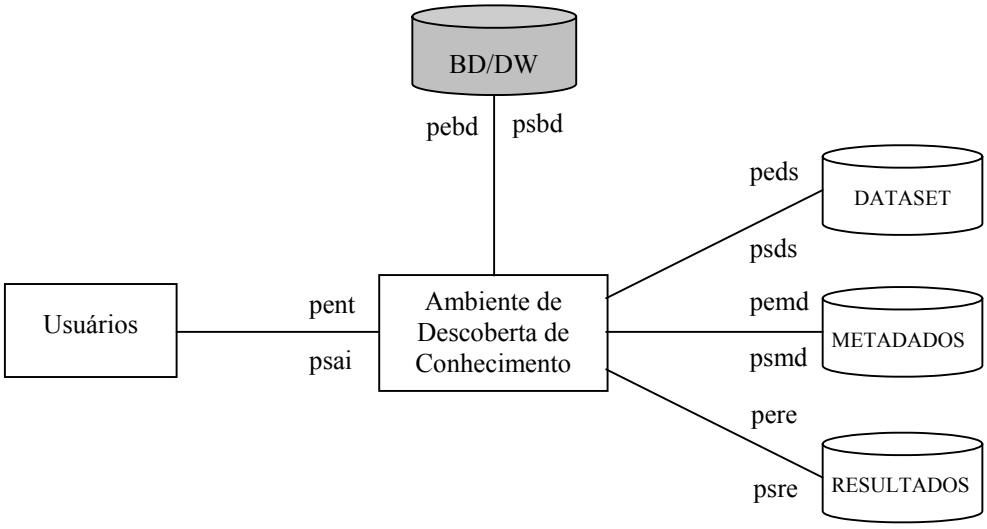


Figura 4.7: Modelo Geral de Comunicação

O ambiente de descoberta de conhecimento solicita ao usuário os parâmetros necessários, tais como: bancos de dados a serem pesquisados, tipo de técnica de mineração de dados, condições de pesquisa etc.

Para realizar as tarefas solicitadas pelo usuário, o ambiente de descoberta de conhecimento necessita ter acesso a bancos de dados operacionais ou *data warehouse* e a tabelas do *DataSet*, do *Metadados* e de *Resultados*. Esse acesso ocorre através das seguintes portas de comunicação, respectivamente: *pebd* e *psbd*; *peds* e *psds*; *pemd* e *psmd*; *pere* e *psre*.

Os dados do banco de dados/*data warehouse* são usados como base na aplicação de técnicas de mineração de dados.

O ambiente faz a preparação dos dados com a ajuda do usuário na seleção de atributos e na definição das transformações dos dados. Os dados, após sofrerem as transformações necessárias, são armazenados no *DataSet* para, posteriormente, servirem como fontes de entrada na aplicação de técnicas de mineração de dados

Os dados sobre os atributos selecionados e sobre as transformações necessárias são armazenados no *Metadados*.

Os resultados obtidos com a aplicação de uma ou mais técnicas de mineração de dados são armazenados e mostrados ao usuário final que, por sua vez, poderá utilizá-los como resultados finais da pesquisa ou como base para novas pesquisas.

A partir do modelo geral de comunicação apresentado na Figura 4.7, o projetista pode construir o modelo completo de comunicação do sistema detalhando o módulo Ambiente de Descoberta de Conhecimento.

4.6.4 Declaração dos módulos

O comportamento dos objetos é definido por módulos E-LOTOS. A declaração de módulos é feita a partir dos diagramas de estados e de colaboração. Assim, cada diagrama de estados dá origem a um módulo e os diagramas de colaboração fornecem informações necessárias para a definição das sincronizações existentes entre os processos.

Na passagem de UML para E-LOTOS, as classes de objetos são declaradas como módulos, os estados do diagrama de estados, que são as operações/métodos dos objetos, são representados como processos ou funções e as mensagens como ações.

4.6.5 Exemplos

A seguir são relacionados alguns exemplos do mapeamento de diagramas UML para E-LOTOS.

a) Herança

A Figura 4.8 mostra um exemplo de herança simples entre classes de objetos. Esta figura é baseada em UML (Booch et al, 1999).

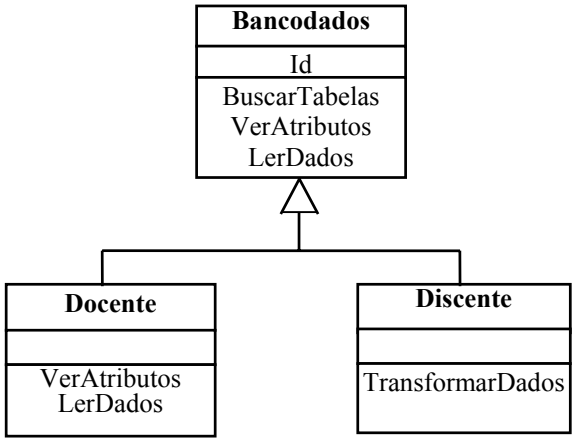


Figura 4.8: Exemplo de Herança entre Classes de Objetos

O exemplo descreve a relação de compartilhamento de atributos e métodos de docentes e discentes, quando ambos são abstraídos como indivíduos.

A especificação na linguagem E-LOTOS do exemplo acima pode ser feita da seguinte forma:

```
module modBancoDados is
  process Bancodados [...] is
        ...
  endproc
  process BuscarTabelas [...] (...) is
        ...
  endproc
  (* ... outros processos *)
endmod
module modDocente import modBancoDados is
  process Docente [...] (...) is
        ...
  endproc
  process VerAtributos [...] (...) is
        ...
  endproc
   (* ... outros processos *)
endmod
module modDiscente import modBancoDados is
  process Discente [...] (...)  is
        ...
  endproc
  process TransformarDados [...] is
        ...
   endproc
   (* ... outros processos *)
endmod
```

A cláusula "*import"* é utilizada para declarar que um módulo herda as características de outro(s) módulo(s).

Cada classe de objetos representada na Figura 4.8 é traduzida para E-LOTOS como um módulo, por exemplo, *modBancoDados* é a especificação da classe de objetos *BancoDados*.

Os processos *BancoDados*, *Docente* e *Discente* representam os construtores das classes e é através deles que ocorre a sincronização com outras classes de objetos.

b) Modelo de comunicação

O modelo de comunicação, apresentado na Figura 4.7, é descrito em E-LOTOS pelos módulos abaixo. O módulo *modDADOS* define os tipos de dados do sistema e o módulo de especificação *AmbDesco* descreve o comportamento do sistema.

```
module modDADOS is
  type Typent is
        ...
  endtype
  type Typesai is
        ...
  endtype
  (* ... outros tipos *)
  type Portaent is (ent: Typent) endtype
  type Portasai is (sai: Typesai) endtype
  type Portaebd is (ebd: Typebd) endtype
  type Portasbd is (sbd: Typebd) endtype
  type Portaeds is (eds: Typeds) endtype
  type Portasds is (sds: Typeds) enddtype
  type Portaemd is (emd: Typemd) endtype
  type Portasmd is (smd: Typemd) endtype
  type Portaere is (ere: Typere) endtype
  type Portasre is (sre: Typere) endtype
endmod
specification AmbDesco import modDados, modUsuario, modAmbiente, modBaseDados,
                              modDataSet, modMetadados, modResultados is
  gates pent: Portaent, psai: Portasai, pebd: Portabd, psbd: Portabd, peds: Portads,
        psds: Portads, pemd: Portamd, psmd: Portamd, pere: Portare, psre: Portare
Behaviour
(
   (
        Usuario [pent, psai]  (0)
        |||
        ...
        Usuario [pent, psai] (n-1)
   )
   |[pent, psai]|
   Ambiente [pent, psai, pebd, psbd, peds, psds, pemd, psmd, pere, psre]
   |[pebd, psbd, peds, psds, pemd, psmd, pere, psre]|
   (
        Bancodados [pebd, psbd]
        |||
        Dataset [peds, psds]
        |||
        Metadados [pemd, psmd]
        |||
        Resultados [pere, psre]
   )
)
endspec
```

O módulo de especificação *AmbDesco* inclui (usando a cláusula *import*) o módulo *modDADOS*, que define os tipos de dados, e os módulos que declaram as classes de objetos *Usuario*, *Ambiente*, *Bancodados*, *Dataset*, *Metadados* e *Resultados*. Ele declara o conjunto de portas tipadas por onde ocorrem as interações entre os objetos e, também, representa os objetos da Figura 4.7 como processos paralelos, que são sincronizados por meio de portas de comunicação.

O conjunto de *n* usuários (indexado de 0 a *n*-1) é sincronizado com o processo *Ambiente*, representando a associação muitos-para-um. Da mesma forma poderão ser representadas as associações um-para-muitos, muitos-para-muitos e um-para-um.

O conjunto de *n* usuários corresponde à instanciação de *n* objetos da classe *Usuario*. Portanto, a instanciação de um objeto em E-LOTOS é representada pela instanciação de um processo (ou um conjunto de processos) que define o estado e o comportamento da classe a qual ele pertence.

c) Definição de módulos

Um módulo deve ser declarado para cada uma das classes de objetos do sistema. A seguir tem-se, como exemplo, o diagrama de estados da classe de objetos Docente, representada na Figura 4.9.

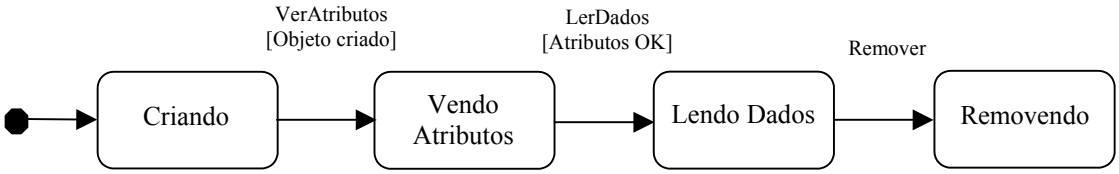


Figura 4.9: Diagrama de Estados para a Classe Docente

O diagrama de estados da figura acima representa os estados da classe de objetos *Docente* e os eventos que causam as mudanças de seus estados.

A especificação na linguagem E-LOTOS do exemplo acima pode ser feita da seguinte forma:

```
module modDocente import modBasedados, modDados is
  function CriarBD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Docente” ;
            true
        endvar
  endfunc
  function RemoverBD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
```

```
            true
        endvar
  endfunc
  process Docente [pebd: Portabd, psbd: Portabd] is
        VerAtributos [pebd, psbd]
     []
        LerDados [pebd, psbd]
  process VerAtributos [pebd: Portabd, psbd: Portabd] is
        var id: integer ;
            infbd: Typebd ;
            atributos: Typeatrib
        in
            pebd (?id, ?infbd) ;
        (* identifica os atributos da tabela Docente *)
            psbd (!id, !atributos) ;
        endvar
  endproc
  process LerDados [pebd: Portabd, psbd: Portabd] is
        var id: integer ;
            fim: boolean ;
            atributos: Typeatrib ;
            dados: Typedados
        in
            pebd (?id, ?atributos) ;
            loop
            (* lê os dados da tabela Docente até o fim *)
                if (fim) then
                  break else
                  psre (!id, !dados) ;
                endif
            endloop
        endvar
  endproc
endmod
```

4.7 IMPLEMENTAÇÃO DO SISTEMA

Nesta etapa, o sistema é implementado a partir dos diagramas UML resultantes da fase de projeto informal, podendo ser utilizado o ambiente ADesC, especificado no Capítulo 5, para facilitar a implementação do sistema. No entanto, o usuário pode optar por implementá-lo em outro ambiente de programação de sua preferência ou utilizando alguma outra ferramenta disponível no mercado (ver Tabela 2.3 no Capítulo 2).

4.8 ANÁLISE DOS RESULTADOS

Nesta fase, são definidas estratégias para a análise dos resultados obtidos pelo sistema de descoberta de conhecimento em banco de dados.

Uma estratégia a ser utilizada é o "Refinamento do Conjunto de Regras Descoberto" (Freitas, 1998), cujo objetivo é reduzir o número de regras (e o número de condições em cada regra) a ser apresentado ao usuário. Freitas (1998) define duas abordagens básicas para esse tipo de estratégia: subjetiva e objetiva.

A abordagem subjetiva (controlada pelo usuário) procura selecionar regras de classificação mais interessantes, baseando-se no conhecimento do usuário. Um exemplo desta abordagem consiste em usar impressões gerais do usuário sobre o domínio de aplicação e então selecionar como interessantes regras que de alguma forma contradizem as impressões gerais do usuário. Assim, são selecionadas as regras que são potencialmente surpreendentes e novas para o usuário . Outro exemplo desta abordagem é o uso de gabaritos de regras para indicar quais atributos podem ocorrer no antecedente e no conseqüente de uma regra (principalmente em regras de associação).

A abordagem objetiva (autônoma, controlada pelo sistema) também procura selecionar regras mais interessantes, mas baseando-se nos dados, em vez de basear-se no conhecimento do usuário. Como exemplo desta abordagem pode-se levar em consideração relacionamentos de inclusão entre diferentes regras.

Nesta etapa, o modelo de mineração de dados construído deve ser validado. Isto implica na avaliação de seus resultados e na interpretação de seu significado.

4.9 CONSIDERAÇÕES FINAIS

O desenvolvimento e a implementação de sistemas de descoberta de conhecimento em banco de dados são tarefas cuja complexidade depende da natureza de seus sistemas e da necessidade do alto grau de conhecimento do negócio da empresa pelo analista de sistemas.

A característica de indeterminismo presente em sistemas de descoberta de conhecimento faz com esses sistemas se diferenciem de outros tipos de sistemas. Por isto, o uso indiscriminado de metodologias clássicas de desenvolvimento de sistemas torna-se inadequado. É necessária a definição de metodologia específica.

A má especificação de qualquer tipo de produto de *software* pode levar a resultados incorretos, que podem causar graves conseqüências. No caso de sistemas de descoberta de conhecimento em banco de dados, os resultados incorretos podem levar a tomadas de decisões também incorretas. Essas decisões podem causar grandes prejuízos

financeiros à empresa, ou um diagnóstico errado (ex: no caso da aplicação de técnicas de mineração de dados na área da saúde).

A metodologia MeDesC define um processo completo e sistemático de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados que aplicam técnicas de mineração de dados.

As etapas da metodologia MeDesC, se seguidas corretamente, levam a especificações de sistemas de descoberta de conhecimento em banco de dados corretas, verificadas e validadas, contribuindo, assim, na construção de sistemas confiáveis e de qualidade.

A metodologia MeDesC e o ambiente ADesC incluem a etapa de projeto formal no ciclo de vida clássico de desenvolvimento de sistemas computacionais, com o objetivo de tornar mais rigoroso o processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Na metodologia MeDesC, os modelos UML são mapeados para a linguagem de especificação formal E-LOTOS. No ambiente ADesC, os modelos UML, que representam as características específicas do sistema projetado, também são traduzidos para E-LOTOS. A linguagem E-LOTOS foi escolhida para ser usada na fase de projeto formal por apresentar características que facilitam a formalização dos modelos UML.

O ambiente ADesC baseia-se na tecnologia de agentes inteligentes para facilitar o desempenho de suas tarefas.

O próximo capítulo apresenta a especificação detalhada do ambiente ADesC.

5 AMBIENTE DE IMPLEMENTAÇÃO DE SISTEMAS DE DESCOBERTA DE CONHECIMENTO EM BANCO DE DADOS

5.1 INTRODUÇÃO

A utilização de sistemas de descoberta de conhecimento em banco de dados para o suporte à tomada de decisão na organização enfrenta, muitas vezes, problemas como: falta de ferramenta adequada; alto custo das ferramentas OLAP e de mineração de dados; e dificuldades no desenvolvimento de um sistema que atenda às necessidades da organização.

As ferramentas disponíveis no mercado freqüentemente atendem a tipos específicos de aplicações. Assim, a organização pode precisar adquirir múltiplas ferramentas para atender as suas necessidades atuais e futuras.

O projeto e implementação de sistemas de descoberta de conhecimento podem constituir-se em tarefas complexas e altamente dependentes da equipe de desenvolvimento. Sua implementação exige do analista de sistemas o conhecimento de tecnologias avançadas da área de informática, tais como: técnicas de mineração de dados, sistemas de gerenciamento de banco de dados, sistemas distribuídos, inteligência artificial, etc. O analista de sistemas também precisa ter conhecimento sobre o negócio da organização. Assim, existe a necessidade de construção de um ambiente que dê suporte ao desenvolvimento desses tipos de sistemas.

Para suprir essa necessidade, foi especificado um ambiente de implementação de sistemas de descoberta de conhecimento em banco de dados, denominado ADesC, apresentado neste capítulo. Este ambiente baseia-se na tecnologia de agentes inteligentes, aplica técnicas de mineração de dados e utiliza diagramas UML para representar suas características. A linguagem E-LOTOS também é usada na especificação formal do ambiente.

5.2 MODELO DO AMBIENTE

O modelo do ambiente ADesC foi construído tendo como base o modelo geral de comunicação, apresentado na Figura 4.7 (Capítulo 4), detalhando o bloco "Ambiente de Descoberta de Conhecimento", como pode ser visto na Figura 5.1.

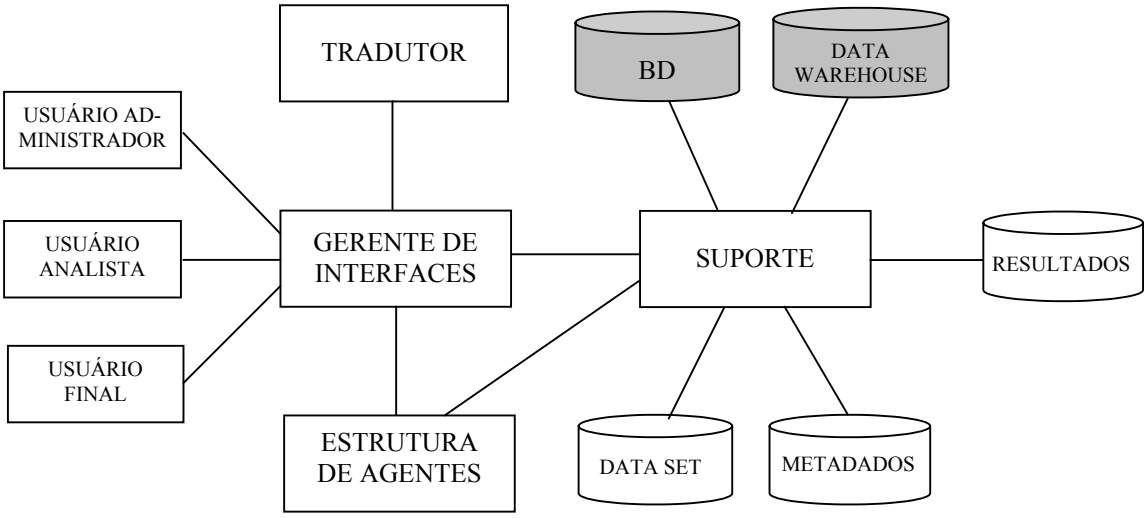


Figura 5.1: Modelo do Ambiente ADesC

Os principais componentes do ambiente ADesC são: Gerente de Interfaces, Suporte, Estrutura de Agentes, Tradutor, Usuário Administrador, Usuário Analista, Usuário Final, Banco de Dados e/ou *Data warehouse*, *Data set*, Metadados e Resultados.

O Usuário Administrador, o Usuário Analista e o Usuário Final representam os tipos de usuários que interagem com o ambiente através do Gerente de Interfaces.

O Usuário Administrador é o responsável pelo cadastramento dos demais usuários do ambiente e pela definição das funções do ambiente e dos bancos de dados e das tabelas dos quais cada usuário poderá ter acesso.

Ao Usuário Analista é atribuída tarefa de desenvolver sistemas de descoberta de conhecimento em banco de dados utilizando a metodologia MeDesC (descrita no Capítulo 4) e a tarefa de entrada dos parâmetros necessários ao ambiente ADesC durante a etapa de implementação do sistema.

O Usuário final, além de auxiliar o analista de sistemas na etapa de análise do sistema, é o responsável pela análise dos resultados.

O Gerente de Interfaces cria e controla todos os objetos de interfaces do ambiente e aqueles utilizados para a visualização dos resultados da execução do sistema de descoberta de conhecimento em banco de dados.

O Suporte é responsável pela identificação dos bancos de dados a serem pesquisados (bancos de dados operacionais e/ou *data warehouse*), pela gravação e acesso dos metadados, pela gravação e acesso dos dados já transformados (*data set*) e pela gravação e acesso dos resultados da aplicação de técnicas de mineração de dados.

O Tradutor traduz modelos UML para a linguagem de especificação formal E-LOTOS.

O Banco de Dados e/ou *Data warehouse* é o repositório dos dados operacionais ou dados multidimensionais usados como fonte de entrada para sistemas de descoberta de conhecimento em banco de dados.

O DataSet é o repositório do conjunto de dados obtido pela função de preparação de dados.

O Metadados é o repositório do catálogo dos dados selecionados das fontes de dados e das transformações definidas como necessárias para a geração do DataSet.

O componente do modelo denominado “Resultados” é o repositório dos resultados obtidos com a aplicação de técnicas de mineração de dados.

A Estrutura de Agentes é responsável pelo gerenciamento dos agentes que compõem o sistema de descoberta de conhecimento em banco de dados. Ela contém os seguintes componentes: Servidor de Agentes, Gerente de Serviços e Coordenador de Transporte. A Figura 5.2 ilustra esta estrutura.

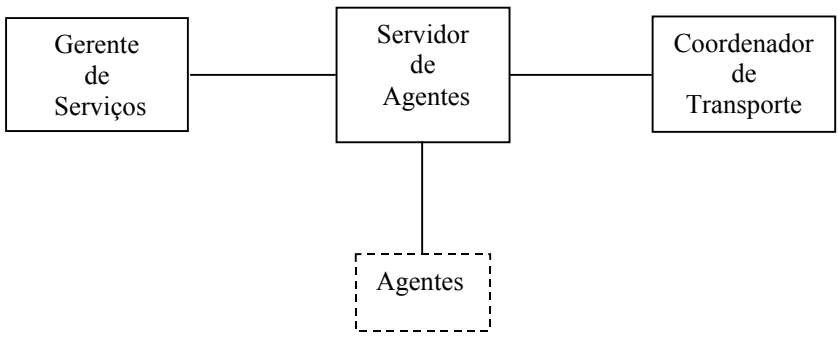


Figura 5.2: Estrutura de Agentes

O Servidor de Agentes é responsável pelo controle do ciclo de vida dos agentes. Para fazer este tipo de controle, ele utiliza serviços oferecidos pelo gerente de serviços, coordenador de transporte, gerente de interfaces e suporte.

O Gerente de Serviços identifica o tipo de serviço solicitado pelo servidor de agente e cria as condições necessárias para que o serviço seja executado.

O Coordenador de Transporte coordena as atividades e os serviços necessários para o transporte de um agente de um nó a outro da rede.

Os tipos de agentes do ambiente ADesC são mostrados na Figura 5.3.

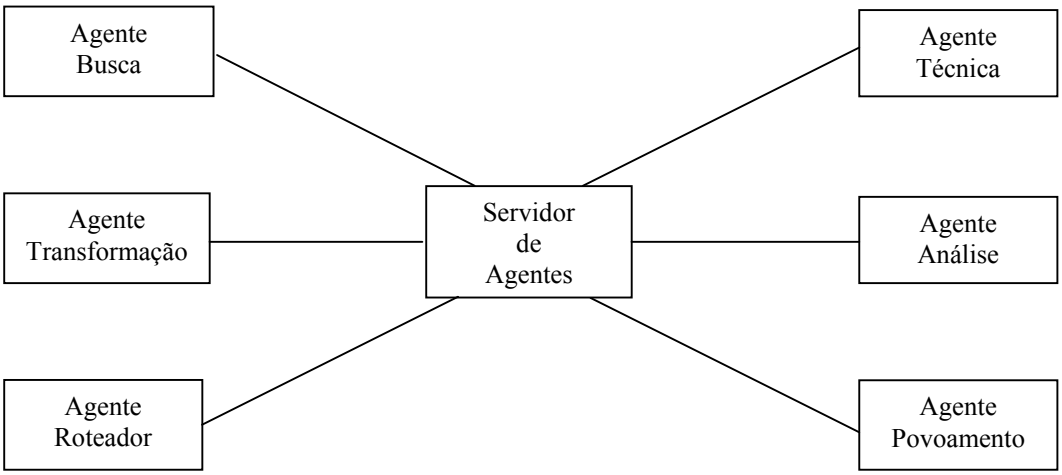


Figura 5.3: Tipos de Agentes

- **Agente busca**: é o agente, móvel ou não, responsável pela busca e seleção de dados nas bases de dados distribuídas e/ou locais.
- **Agente transformação**: é o agente responsável pela transformação dos dados que darão origem ao conjunto de dados a ser utilizado na aplicação da técnica de mineração de dados.
- **Agente roteador**: é o agente responsável pela definição da rota que um agente de busca móvel deverá seguir para chegar ao nó destino da rede.
- **Agente técnica**: é o agente que implementa uma técnica de mineração de dados, desempenhando uma das seguintes tarefas: classificação, estimativa, associação, segmentação e sumarização. Para cada uma das ténicas de mineração de dados deve ser implementado um agente desse tipo.
- **Agente análise**: é o agente que auxilia o usuário na análise dos resultados.
- **Agente povoamento**: é o agente responsável pelo povoamento do DataSet.

Os agentes são projetados como objetos. Além dos métodos que implementam as funções específicas estabelecidas a cada agente, um agente tem, também, os seguintes métodos:

- **Criação**: um novo agente é criado com um identificador, inserido em um contexto e inicializado.
- **Clonagem**: a clonagem de um agente produz uma cópia idêntica do agente original no mesmo contexto; o novo agente é assinalado com um novo identificador e a execução reinicia nele.
- **Expedição**: um agente é despachado de um contexto para outro onde ele reinicia sua execução.
- **Retração**: a retração remove um agente de seu contexto atual e o insere no contexto no qual a retração foi requerida.
- **Desativação**: a desativação de um agente é a habilidade de remover temporariamente um agente de seu contexto atual e armazená-lo na memória secundária.
- **Ativação**: a ativação devolve um agente desativado ao seu contexto.
- **Remoção**: a remoção pára a execução atual de um agente e o remove de seu contexto atual.

Os principais serviços são os seguintes:

- **Nomeação**: nomeia agentes para que eles possam ser distingüidos entre si.
- **Migração**: este serviço controla como a transferência de um agente é realizada entre nós da rede, empacotando o agente dentro de um formato adequado para a transmissão na rede.
- **Aquisição de dados**: agentes móveis interrogam seu ambiente local para adquirir a informação necessária para cumprir seus objetivos.
- **Determinação de rota**: uma rota deve ser determinada para o agente seguir até o seu destino, sendo que esta pode ser pré-determinada, ter determinação dinâmica ou determinação híbrida.
- **Comunicação**: a comunicação entre agentes envolve enviar, receber e manipular mensagens de forma síncrona bem como assíncrona; podem existir dois métodos para comunicação de agentes: orientado à rede e orientado ao nó.
- **Espera**: os agentes são armazenados enquanto esperam por longos períodos de tempo para que eventos ocorram.

- **Persistência**: os dados e o estado de execução do agente são armazenados.
- **Segurança**: as ações necessárias de segurança são realizadas antes, durante e após o transporte.
- **Serviço de tolerância a falhas**: para evitar que informações sejam perdidas ou alteradas devido a problemas na rede ou nos nós da rede, um agente deve ser protegido de algumas situações que podem ocorrer em seu tempo de vida, tais como:
  - **falha de transmissão na rede**: se um erro ocorre enquanto o agente está em trânsito entre os nós, então ele deve ser retransmitido ou reiniciado no nó de onde ele foi transmitido;
  - **falha no nó**: se o agente está em execução em um nó que falha, então deve ser possível mandar o agente de volta e recuperar uma cópia sua;
  - **violação de segurança**: se um agente não pode ser transmitido a uma nova localização, devido a uma restrição de segurança (por exemplo, o agente não é do nó verdadeiro); ele deve ser reinicializado no nó local e dada uma chance de escolher um novo destino.

5.3 PROJETO INFORMAL DO AMBIENTE

O projeto informal do ambiente ADesC baseia-se na tecnologia de agentes inteligentes para a definição das classes de objetos necessárias e utiliza diagramas UML para a representação dos objetos e de seus comportamentos.

A seguir são apresentados os casos de uso do ambiente, as classes de objetos com seus atributos e métodos, as funções desempenhadas pelo ambiente, o projeto da interface e o modelo de metadados.

5.3.1 Casos de Uso do Ambiente

Os diagramas casos de usos representam a interação entre o sistema e seus usuários. Os casos de uso definidos para o ambiente estão agrupados pelos três tipos de usuários previstos, são eles: usuário administrador, usuário analista e usuário final. O usuário administrador tem acesso a todas as funções do sistema e é responsável pela configuração do sistema; ou seja, cadastramento de usuários e definição dos tipos de acesso de cada usuário. As tarefas de desenvolvimento e implementação de sistemas de

descoberta de conhecimento em banco de dados são atribuídas ao usuário analista. O usuário final tem acesso apenas aos resultados obtidos pelo sistema e é responsável pela análise desses resultados. As Figuras 5.4 e 5.5 mostram os diagramas casos de uso do usuário analista e do usuário final, respectivamente. O diagrama caso de uso do usuário administrador não está aqui representado porque, além da função de "Configuração", este tipo de usuário tem acesso às mesmas funções do usuário analista.

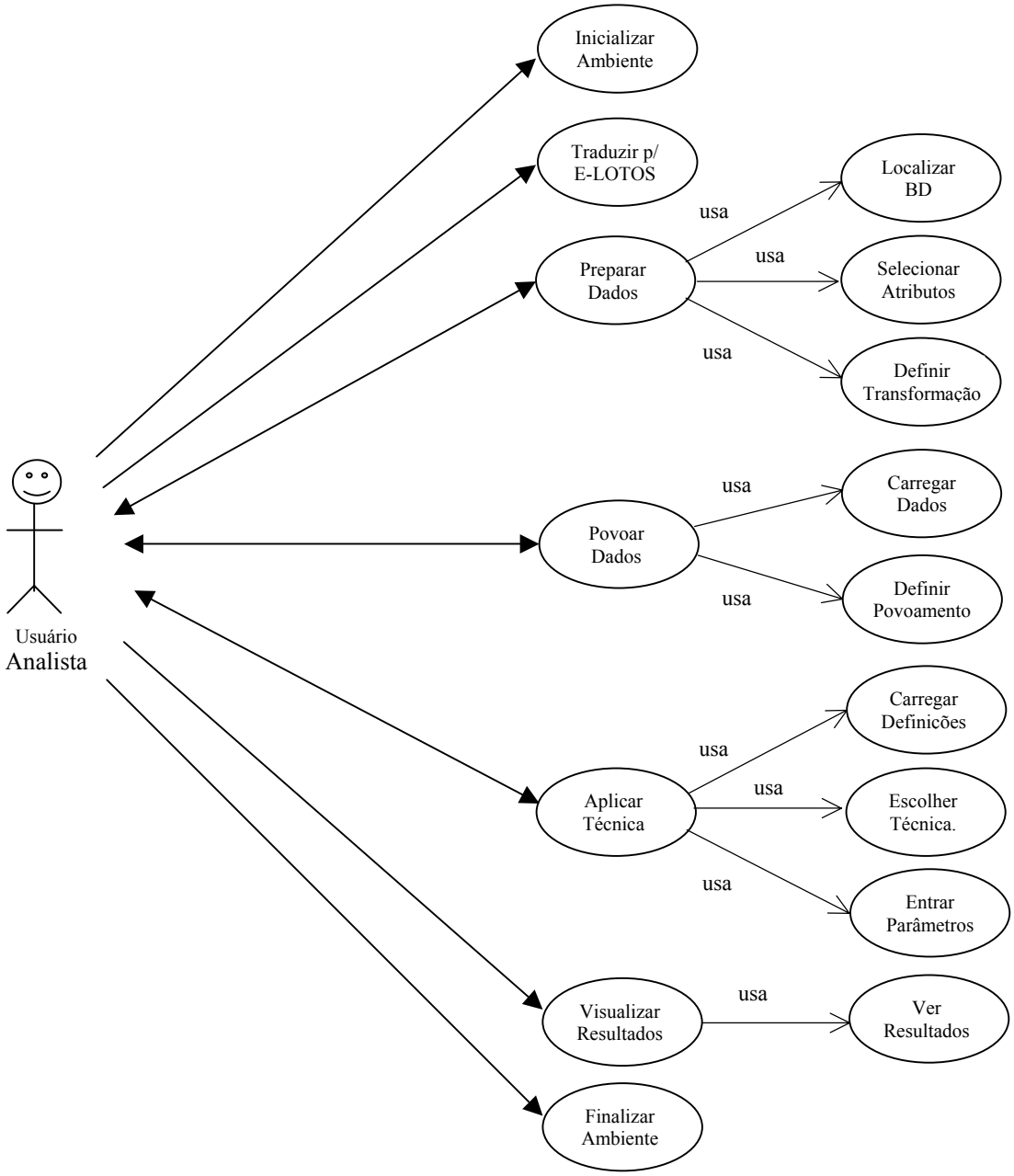


Figura 5.4: Diagrama Caso de Uso – Usuário Analista

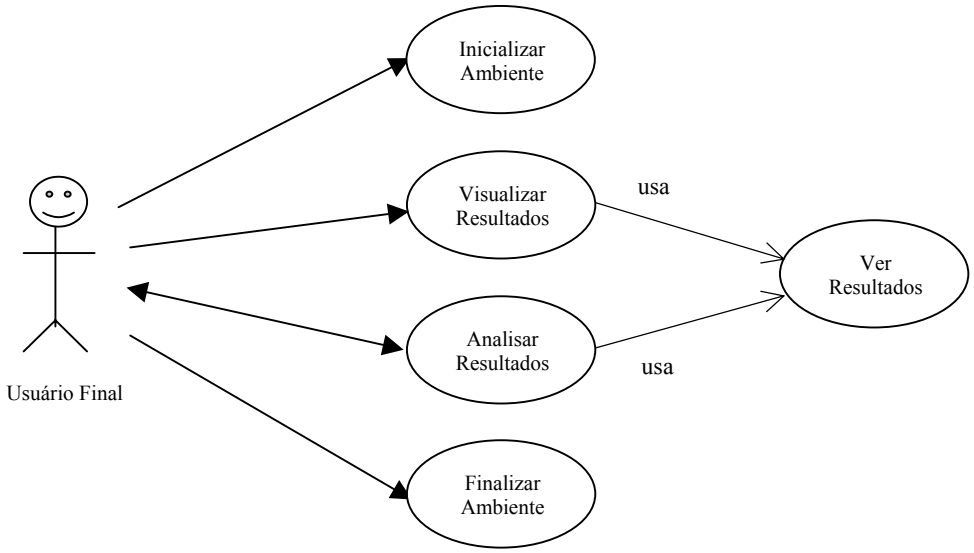


Figura 5.5: Diagrama Caso de Uso – Usuário Final

5.3.2 Definição das Classes de Objetos

A Tabela 5.1 relaciona as classes de objetos com seus métodos e atributos e a Figura 5.6 mostra o diagrama de classes do ambiente.

5.3.3 Funções do Ambiente

As principais funções do ambiente ADesC são: Inicialização, Configuração, Tradução para E-LOTOS, Preparação de Dados, Povoamento, Mineração de Dados, Visualização dos Resultados, Análise dos Resultados e Finalização, que são descritas na seqüência.

A ordem de execução dessas funções depende do tipo de usuário e das funções já realizadas anteriormente. O Usuário Administrador executa, normalmente nesta ordem, as funções de Inicialização, Configuração e Finalização. O Usuário Analista pode executar todas as funções, exceto Configuração, na ordem em que estão relacionadas ou deixar de executar alguma delas se o objetivo naquele momento é, por exemplo, manter atualizado o *data set* ou aplicar a técnica de mineração de dados com novos parâmetros.

Tabela 5.1: Classes de Objetos do Ambiente

CLASSES	ATRIBUTOS	MÉTODOS
Suporte	IdBD, NomeBD, Endereço	Criar, Remover, LocalizarBD, VerAtributos, GravarDefinições, GravarDados, CarregarDados, GravarResultados, CarregarResultados, CarregarMetadados, AtualizarDados, RemoverTabelas
GerenteInterfaces	TipoInterface	Criar, Remover, InicializarAmbiente, PrepararDados, MostrarAtributos, AplicarTécnica, VisualizarResultados, AnalisarResultados, FinalizarAmbiente
InterfaceEntrada	IdBD, NomeBD, Endereço	Criar, Remover, SelecionarBD
InterfaceDados	IdTabela, NomeTabela, NomeAtributo	Criar, Remover, MostrarAtributos, SelecionarAtributos
InterfaceTransformação	IdTabela, NomeTabela, NomeAtributo, TipoTransf, Expressão	Criar, Remover, DefinirTransformação
InterfaceTécnica	IdDS, NomeDS, IdTécnica, IdAlgoritmo, Parâmetros	Criar, Remover, CarregarDefinições EscolherTécnica, EntrarParâmetros
InterfaceResultados	IdResul, IdTécnica, IdAlgoritmo, Resultados	Criar, Remover, MostrarResultados
InterfaceAnálise	IdResul, IdTécnica, IdAlgoritmo, DadosAnálise	Criar, Remover, MostrarAnálise
InterfacePovoamento	IdDS, IdTabela, TipoEvento, DadosEvento	Criar, Remover, CarregarDados, DefinirPovoamento
ServidorAgentes	TipoAgente	Criar, Remover, DefinirAtributos, TransformarDados, DefinirAlgoritmo, AplicarAlgoritmo, RealizarAnálise, RemoverAgentes
AgenteBusca	IdBD, NomeBD, Endereço, Tabelas (IdTabela, NomeTabela)	Criar, Remover, BuscarTabelas
AgenteRoteador	IdRota, Endereço	Criar, Remover, DefinirRota
AgenteTransformação	IdTabela, NomeTabela, NomeAtributo, TipoTransf, Expressão	Criar, Remover, TransformarDados
AgenteTécnica	IdTécnica, IdAlgoritmo, Parâmetros	Criar, Remover, DefinirAlgoritmo, AplicarAlgoritmo
AgenteAnálise	IdTécnica, IdAlgoritmo, Resultados	Criar, Remover, Analisar
AgentePovoamento	IdBD, IdTabela, TipoEvento, DadosEvento	Criar, Remover, Povoar
GerenteServiços	TipoServiço	Criar, Remover, VerServiço, AtivarServiço, NomearAgente, ControlarMigração, AdquirirDados, DeterminarRota, ControlarComunicação, TratarEspera, TratarPersistência, GarantirSegurança, EvitarFalhas
CoordenadorTransporte	IdBD, Endereço	Criar, Remover, CoordenarTransporte
Usuário	Tipo, IdUsuário, Senha	Criar, Remover, VerTipo, VerificarSenha
BancoDados/ DataWarehouse	IdBD, NomeBD, Endereço, ListaTabelas	Criar, Remover, VerAtributos, LerBancoDados
DataSet	IdDS, NomeDS, Dados	Criar, Remover, GravarDataSet, LerDataSet
Metadados	IdBD, IdTabela, Atributo, TipoTransf, Expressão	Criar, Remover, GravarDefinições, LerMetadados, Atualizar
Resultados	IdResul, IdTécnica, IdAlgoritmo, Parâmetros	Criar, Remover, Gravar, LerResultados

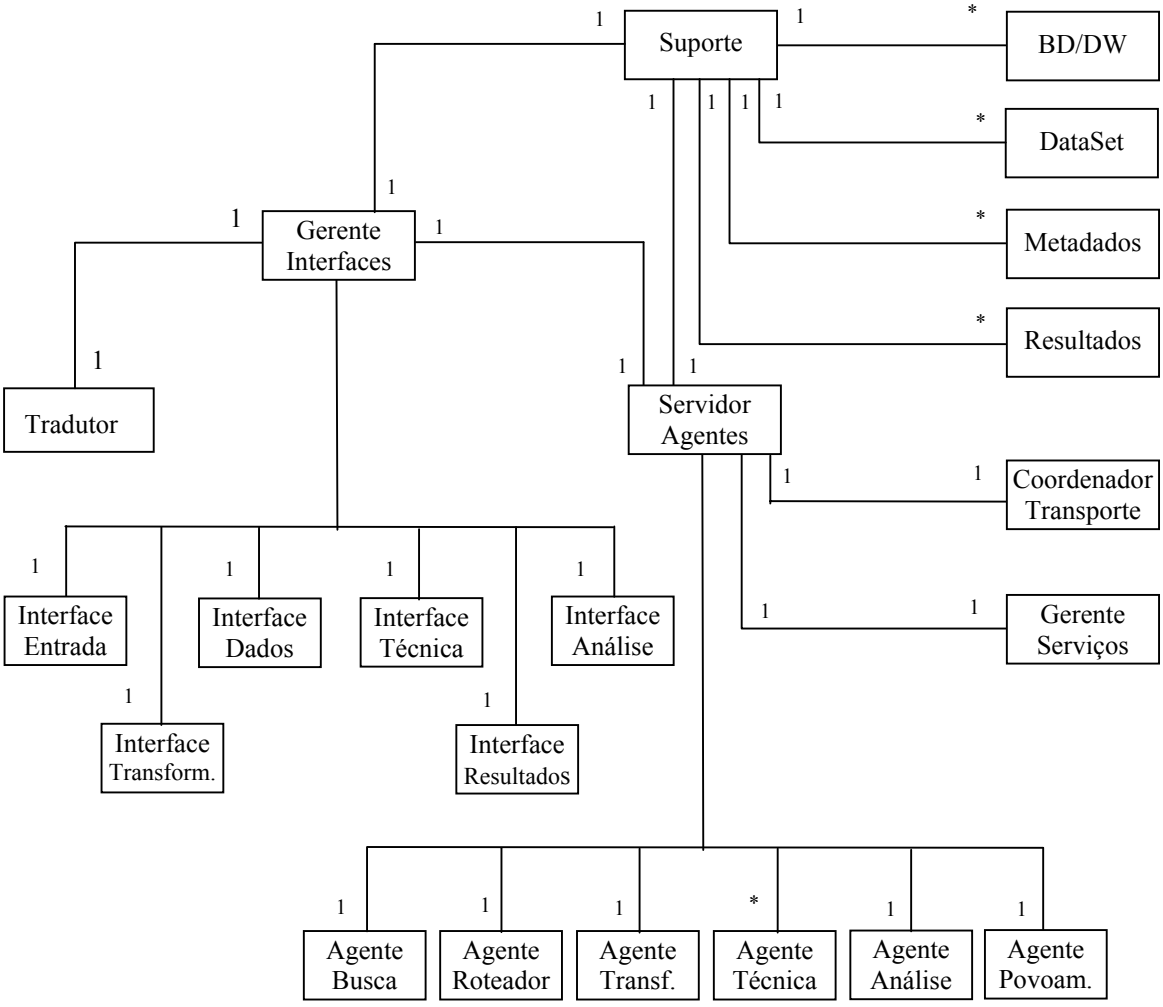


Figura 5.6: Diagrama de Classes do Ambiente

1) Inicialização

A inicialização do ambiente ocorre no momento em que o programa de execução do ambiente é disparado por um usuário. Esta função cria os principais objetos do sistema, são eles: GerenteInterfaces, Suporte e ServidorAgente. O Diagrama de Colaboração da Figura 5.7 ilustra esta função.

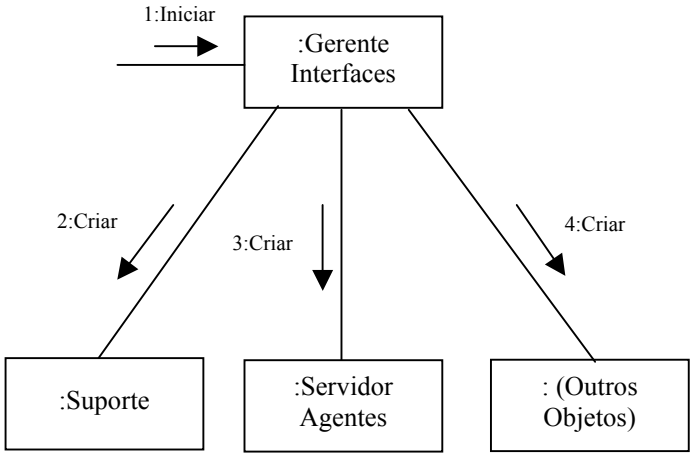


Figura 5.7: Diagrama de Colaboração – Inicializar Ambiente

2) Configuração

Através desta função, o Usuário Administrador cadastra os demais usuários do ambiente e define quais funções devem ficar visíveis a cada um deles de acordo com a classe de usuário a que ele pertence. A Figura 5.8 representa esta função.

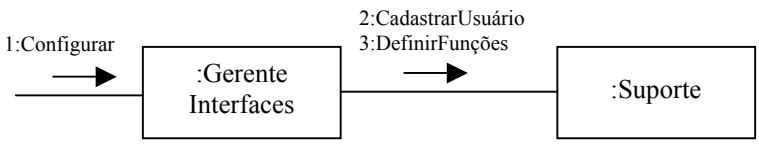


Figura 5.8: Diagrama de Colaboração – Configurar

3) Tradução para E-LOTOS

A tradução dos modelos UML para E-LOTOS é realizada automaticamente pelo ambiente de acordo com o que foi definido como forma de mapeamento no Capítulo 4.

4) Preparação de dados

A função de preparação de dados envolve os passos a seguir e o diagrama de colaboração da Figura 5.9 ilustra esta função.

- Criação dos objetos necessários para a execução desta função pelo ambiente.
- Entrada da identificação do(s) banco(s) de dados/*data warehouse* a ser(em) pesquisado(s) e sua localização pelo usuário analista e através da interface entrada.
- Localização do(s) banco(s) de dados/*data warehouse* de entrada para o sistema, pelo agente busca e com a ajuda do Suporte. Quando houver algum banco de dados localizado em outra estação da rede, o Gerente de Serviços e o Coordenador de Transporte farão a ativação dos serviços necessários e o controle de transporte do agente busca, respectivamente.
- Seleção dos atributos mais relevantes ao tipo de estudo que se pretende realizar, pelo usuário analista através da interface dados.
- Definição dos tipos de transformações necessárias aos atributos selecionados, realizada pelo usuário analista e através da interface transformação.
- Gravação dos tipos de transformações no Metadados.
- Leitura dos dados contidos no banco de dados/*data warehouse*, através do Suporte.
- Realização das transformações necessárias nos dados pelo agente transformação.
- Aplicação de técnica(s) estatística(s) nos dados já transformados para obter uma amostra, reduzindo assim o volume de dados; também realizada pelo agente transformação.
- Gravação dos conjuntos de dados selecionados e transformados no DataSet, com a ajuda do suporte.
- Remoção dos objetos criados por esta função.

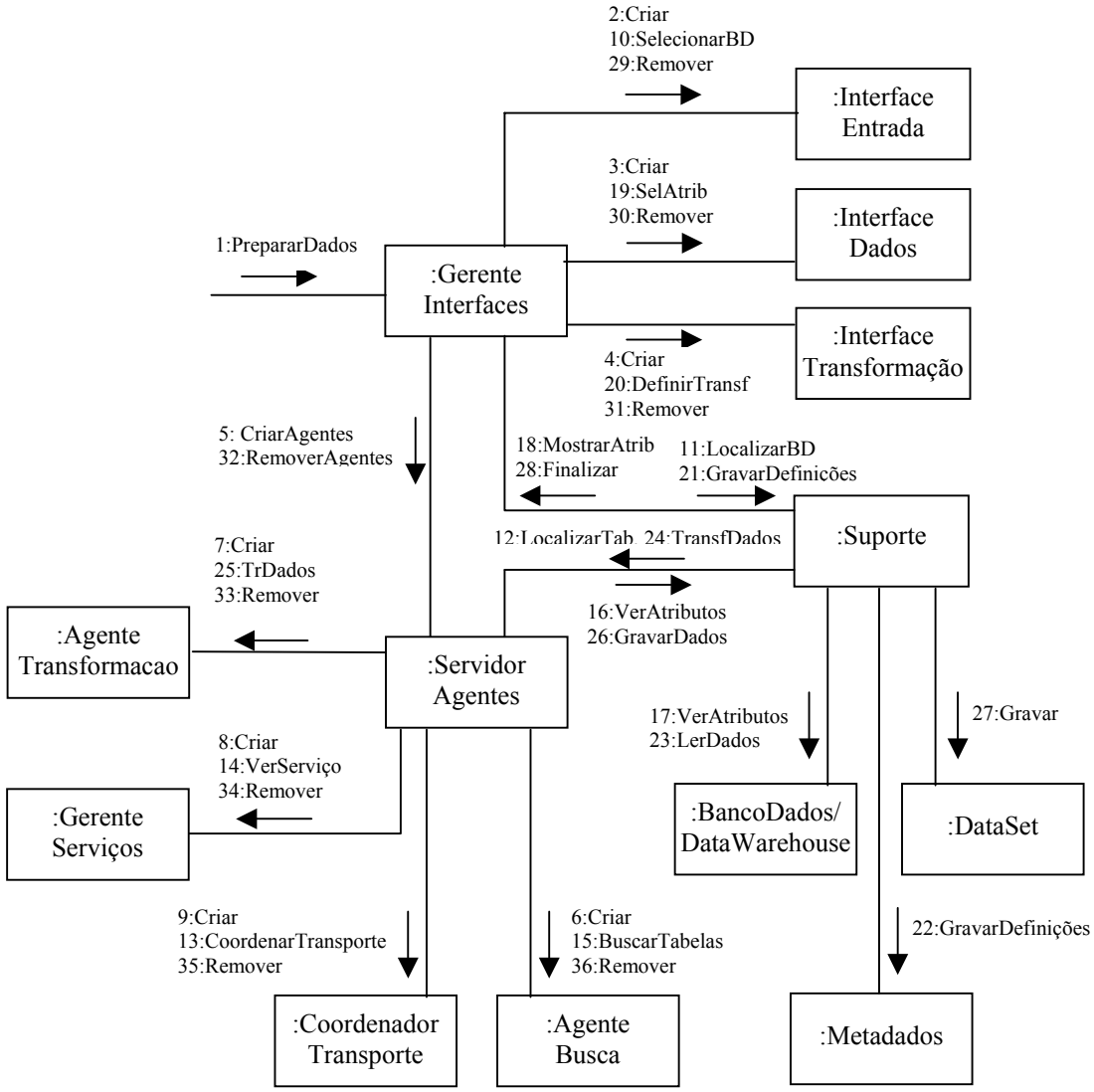


Figura 5.9: Diagrama de Colaboração – Preparar Dados

5) Povoamento

Nesta função, o conjunto de dados do DataSet é povoado com dados atualizados através do Metadados gravado pela função de preparação de dados. O povoamento pode ser ativado através de um evento automático que é disparado pelo sistema ou através de um comando executado pelo usuário analista. O diagrama de colaboração da Figura 5.10 representa esta função e a seguir são relacionados os passos realizados nesta função:

- Criação dos objetos necessários para a execução da função de povoamento.
- Definição da(s) forma(s) de povoamento pelo usuário analista.
- Aplicação do povoamento pelo agente povoamento.

- Atualização do Metadados.
- Gravação de novos dados no DataSet.
- Remoção dos objetos criados.

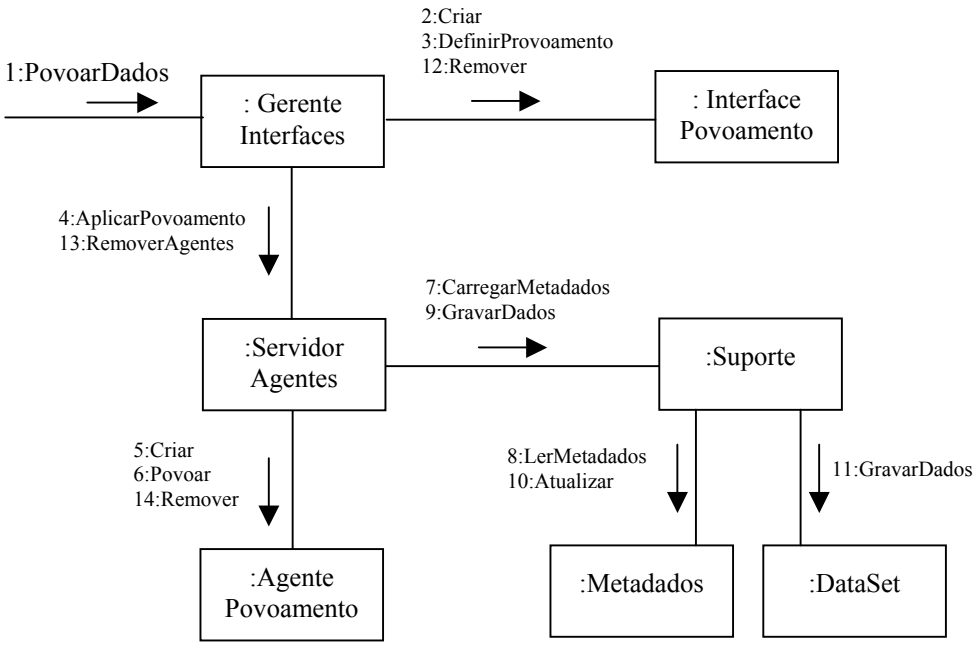


Figura 5.10: Diagrama de Colaboração – Povoamento

6) Mineração de dados

A função de mineração de dados define os passos a seguir e o diagrama de colaboração da Figura 5.11 ilustra esta função.

- Criação dos objetos necessários para a execução da função de mineração de dados, pelo ambiente.
- Escolha de uma técnica de mineração de dados, pelo usuário analista e através da interface técnica.
- Definição do algoritmo a ser utilizado na aplicação da técnica escolhida, pelo agente técnica.
- Entrada dos parâmetros necessários para o algoritmo definido.
- Leitura dos dados do DataSet pelo suporte.
- Aplicação do algoritmo de mineração de dados, pelo agente técnica.
- Gravação dos resultados obtidos, pelo suporte.
- Remoção dos objetos criados.

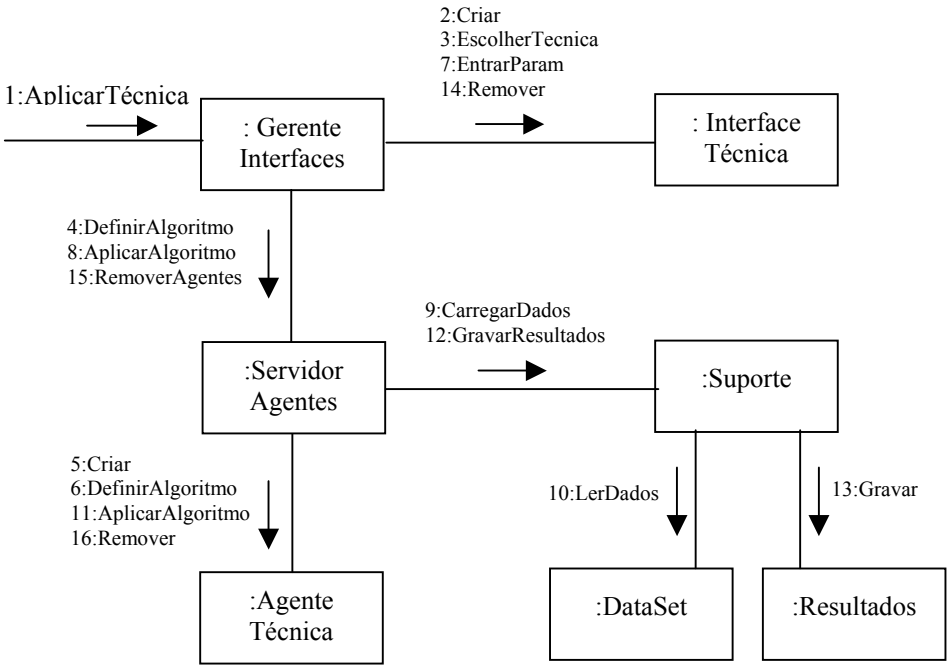


Figura 5.11: Diagrama de Colaboração – Mineração de Dados

7) Visualização dos resultados

A função de visualização dos resultados segue os seguintes passos e o diagrama de colaboração desta função é apresentado na Figura 5.12.

- Criação dos objetos necessários para a execução da função de visualização dos resultados.
- Leitura dos resultados obtidos na função de mineração de dados, pelo suporte.
- Visualização dos resultados através da interface resultados.
- Remoção dos objetos criados.

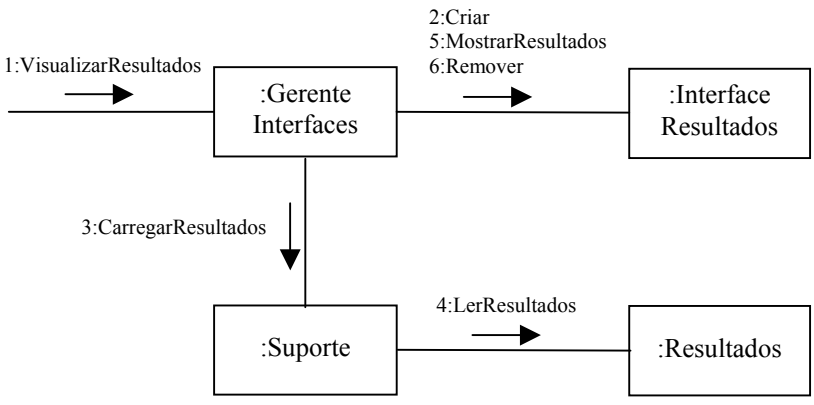


Figura 5.12: Diagrama de Colaboração – Visualizar Resultados

8) Análise dos resultados

Para a análise dos resultados obtidos pelo sistema, são definidos os passos a seguir e o diagrama de colaboração da Figura 5.13 ilustra esta função.

- Criação dos objetos necessários para a execução da função de análise dos resultados.
- Leitura dos resultados obtidos na função de mineração de dados.
- Visualização dos resultados através da interface resultados.
- Realização da análise dos resultados pelo agente análise.
- Visualização da análise através da interface análise.
- Remoção dos objetos criados.

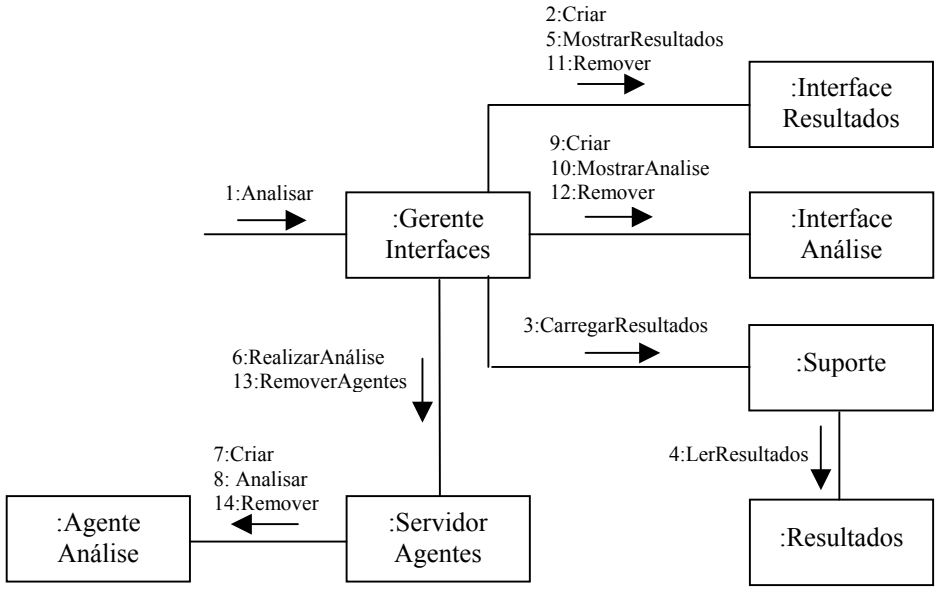


Figura 5.13: Diagrama de Colaboração – Analisar Resultados

9) Finalização

A finalização remove os objetos criados pela inicialização. O diagrama de colaboração da Figura 5.14 ilustra esta função.

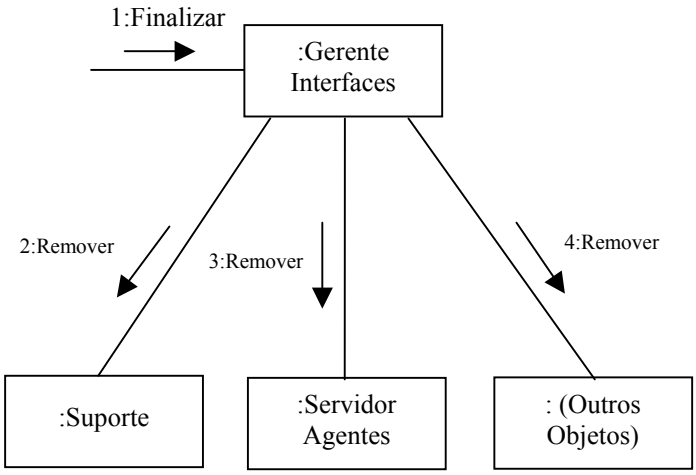


Figura 5.14: Diagrama de Colaboração – Finalizar Ambiente

5.3.4 Diagramas de Estados

Os diagramas de estados são utilizados na especificação informal do ambiente para representar os estados e os eventos que, quando disparados, mudam o estado do objeto e, também, para servirem como base na formalização do ambiente. O Anexo 9.2 mostra esses diagramas.

5.3.5 Estrutura do Metadados

Um metadados é uma estrutura de dados definida para armazenar informações sobre os dados do sistema. Essas informações são referentes aos atributos selecionados das tabelas dos bancos de dados operacionais, às transformações realizadas e à forma de povoamento dos dados no DW/DM/DS.

O Anexo 9.3 apresenta o diagrama entidade-relacionamento do Modelo de Metadados proposto e a descrição do conteúdo desse modelo[1].

5.4 PROJETO FORMAL DO AMBIENTE

No projeto formal do ambiente é realizado o mapeamento de diagramas UML para E-LOTOS, conforme foi definido na metodologia MeDesC, apresentada no

[1] Uma estrutura de metadados está em fase de definição por um aluno de mestrado do Curso de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina.

Capítulo 4. Os diagramas de colaboração e, principalmente, os diagramas de estados são utilizados como base para este mapeamento. A seguir, na Figura 5.15, é mostrado o modelo de comunicação do ambiente ADesC usado como base para a especificação formal. A especificação formal completa do ambiente ADesC na linguagem E-LOTOS pode ser vista no Anexo 9.4.

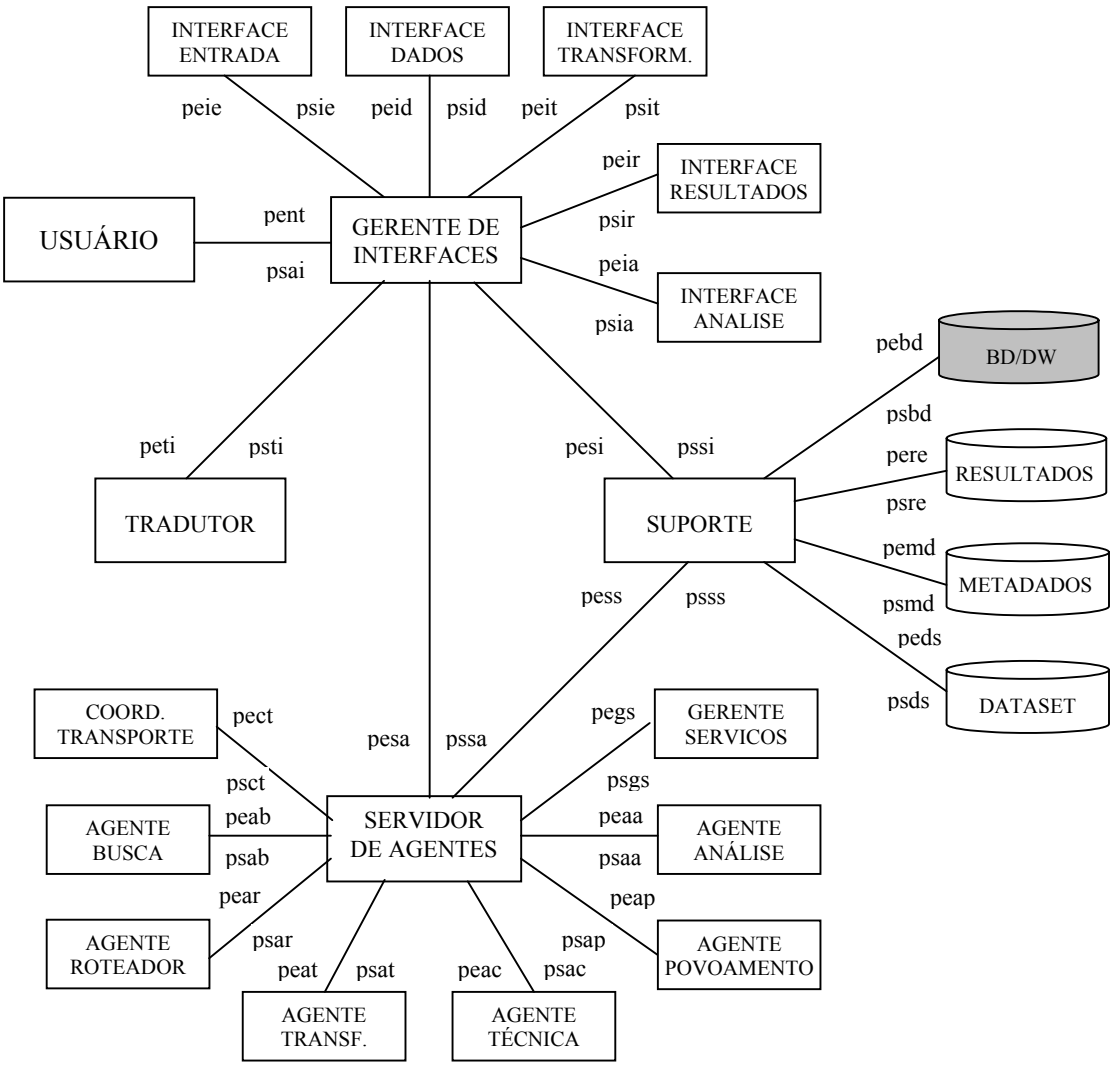


Figura 5.15: Modelo de Comunicação do Ambiente ADesC

5.5 CONSIDERAÇÕES FINAIS

A especificação de um ambiente de implementação de sistemas de descoberta de conhecimento constitui tarefa complexa devido à natureza e ao aspecto heterogêneo desses sistemas e, também, ao nível de detalhamento necessário.

A especificação informal do ambiente ADesC foi realizada baseando-se na tecnologia de agentes inteligentes, por ser um tipo de tecnologia que vem sendo bastante utilizada na solução de problemas computacionais complexos.

A utilização de modelos UML na etapa de projeto informal facilita o entendimento do sistema através da representação gráfica de suas características.

A especificação formal de um sistema constitui, também, em uma tarefa complexa porque toda técnica de descrição formal possui níveis elevados de dificuldades tanto no entendimento quanto na sua utilização. Mas, por outro lado, o uso de uma TDF na especificação de sistemas possibilita a sua verificação e sua validação, tornando os sistemas mais confiáveis e de melhor qualidade.

O ambiente ADesC foi especificado formalmente porque o seu objetivo principal é a implementação de sistemas de descoberta de conhecimento projetados através da metodologia MeDesC, que usa formalismo no processo de desenvolvimento de sistemas.

Com o uso do ambiente ADesC, os modelos UML, que representam as características do sistema de descoberta de conhecimento em banco de dados projetado, podem ser traduzidos automaticamente para a linguagem E-LOTOS e incorporadas ao ambiente, de forma a completar toda a implementação do sistema. Portanto, o analista de sistemas, após especificar o sistema usando a metodologia MeDesC, precisa, basicamente, informar ao ambiente os parâmetros necessários para a preparação de dados (seleção de atributos e transformação de dados) e escolher a técnica de mineração de dados a ser aplicada.

Após a execução de um algoritmo que implementa a técnica de mineração de dados escolhida, o ambiente ADesC retornará os resultados da mineração de dados realizada e ajudará o analista de sistemas e o usuário final na etapa de análise dos resultados, através da implementação de agentes inteligentes.

O Capítulo 6 apresenta a aplicação do modelo proposto através de um estudo de caso que mostra o desenvolvimento de um sistema de descoberta de conhecimento em banco de dados, utilizando a metodologia MeDesC e o ambiente ADesC.

6 APLICAÇÃO DO MODELO

6.1 INTRODUÇÃO

Este capítulo descreve estudo de casos que tem como objetivo principal validar o modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, apresentado no Capítulo 4. Mais especificamente, validar a metodologia MeDesC e o ambiente ADesC (especificado no Capítulo 5).

A validação é realizada com o desenvolvimento de um sistema de descoberta de conhecimento seguindo as etapas da metodologia MeDesC, utilizando o protótipo do ambiente ADesC na etapa de implementação e tendo como base os dados contidos em bancos de dados da CAPES dos Programas de Pós-Graduação do Brasil no ano de 1998.

6.2 ESPECIFICAÇÃO DO SISTEMA

A especificação do sistema, apresentada nas próximas seções, mostra o desenvolvimento de um sistema de descoberta de conhecimento em banco de dados utilizando a metodologia MeDesC descrita no Capítulo 4.

A metodologia MeDesC é seguida etapa por etapa para demonstrar a forma rigorosa e sistemática por ela imposta e para servir como base ao desenvolvimento de outros sistemas de descoberta de conhecimento.

As etapas e sub-etapas, descritas nas próximas seções, são:

1) Análise do Sistema:
   - Descrição do problema;
   - Definição dos objetivos do sistema;
   - Construção de diagramas de classes do(s) sistema(s) em operação;
   - Construção do diagrama de classes corporativo.
2) Projeto Informal:
   - Seleção dos atributos;
   - Definição das transformações dos dados;
   - Projeto de uma estrutura de metadados;
   - Projeto do DW/DM/DS;

- Definição de técnica de amostragem estatística;
- Escolha de técnica de mineração de dados;
- Construção de diagramas de classes;
- Construção de diagramas de estados;
- Construção de diagramas de colaboração;
- Determinação da forma de povoamento do DW/DM/DS;

3) Projeto Formal:
   - Divisão do sistema em módulos;
   - Definição do modelo de comunicação;
   - Declaração dos módulos.
4) Implementação do Sistema.
5) Análise dos Resultados.

6.3 ANÁLISE DO SISTEMA

A etapa de Análise do Sistema está subdividida em quatro sub-etapas, que são descritas a seguir, são elas:

1) Descrição do problema;
2) Definição dos objetivos do sistema;
3) Construção de diagramas de classes do(s) sistema(s) em operação;
4) Construção do diagrama de classes corporativo.

6.3.1 Descrição do Problema

Na descrição do problema de mineração de dados a ser solucionado é dada uma visão geral sobre a área de negócio a ser investigada. Neste estudo de casos, a área a ser investigada é Fomento à Pós-Graduação. A CAPES é uma agência de fomento à Pós-Graduação do Brasil que mantém um sistema de avaliação de cursos, reconhecido e utilizado por outras instituições nacionais.

> "A Fundação de Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) é uma entidade pública vinculada ao Ministério da Educação - MEC, criada inicialmente como Campanha, em 1951 e

> instituída como Fundação em 1992. Ao longo desses anos, a CAPES vem cumprindo seu objetivo principal de subsidiar o MEC na formulação das políticas de pós-graduação, coordenando e estimulando - mediante a concessão de bolsas de estudo, auxílios e outros mecanismos - a formação de recursos humanos altamente qualificados para a docência em grau superior, a pesquisa e o atendimento da demanda profissional dos setores públicos e privados" (CAPES, 2000).

No plano geral, a CAPES tem as seguintes finalidades (CAPES, 2000):

- Elaborar a proposta do Plano Nacional de Pós-Graduação, acompanhar e coordenar a sua execução;
- Elaborar planos de atuação setoriais ou regionais;
- Promover estudos e avaliações necessários ao desempenho de suas atividades;
- Fomentar estudos e atividades que, direta ou indiretamente, contribuam para o desenvolvimento e consolidação das instituições de ensino superior;
- Apoiar o processo de desenvolvimento científico e tecnológico nacional;
- Manter intercâmbio e contato com outros órgãos da Administração Pública ou entidades, privadas, nacionais e internacionais, visando à celebração de convênios, acordos, contratos e ajustes relativos à consecução de seus objetivos.

Para o desempenho de suas atividades, a CAPES utiliza-se de pareceres de consultores científicos, com a finalidade de:

- Proceder ao acompanhamento e à avaliação dos Programas de pós-graduação;
- Apreciar o mérito das solicitações de bolsas ou auxílios;
- Opinar sobre matérias que lhe sejam suscitadas pelo Conselho Técnico-Científico ou pelo seu Presidente.

A CAPES possui bancos de dados contendo informações sobre os Programas de Pós-Graduação do Brasil, tais como: nível do curso, pesquisadores vinculados ao Programa, produção científica, teses/dissertações defendidas, identificação do orientador de cada tese/dissertação, etc. A Figura 6.1 mostra o modelo de dados desses bancos de dados. A descrição detalhada das tabelas que compõem esse modelo pode ser encontrada em (CAPES, 1999).

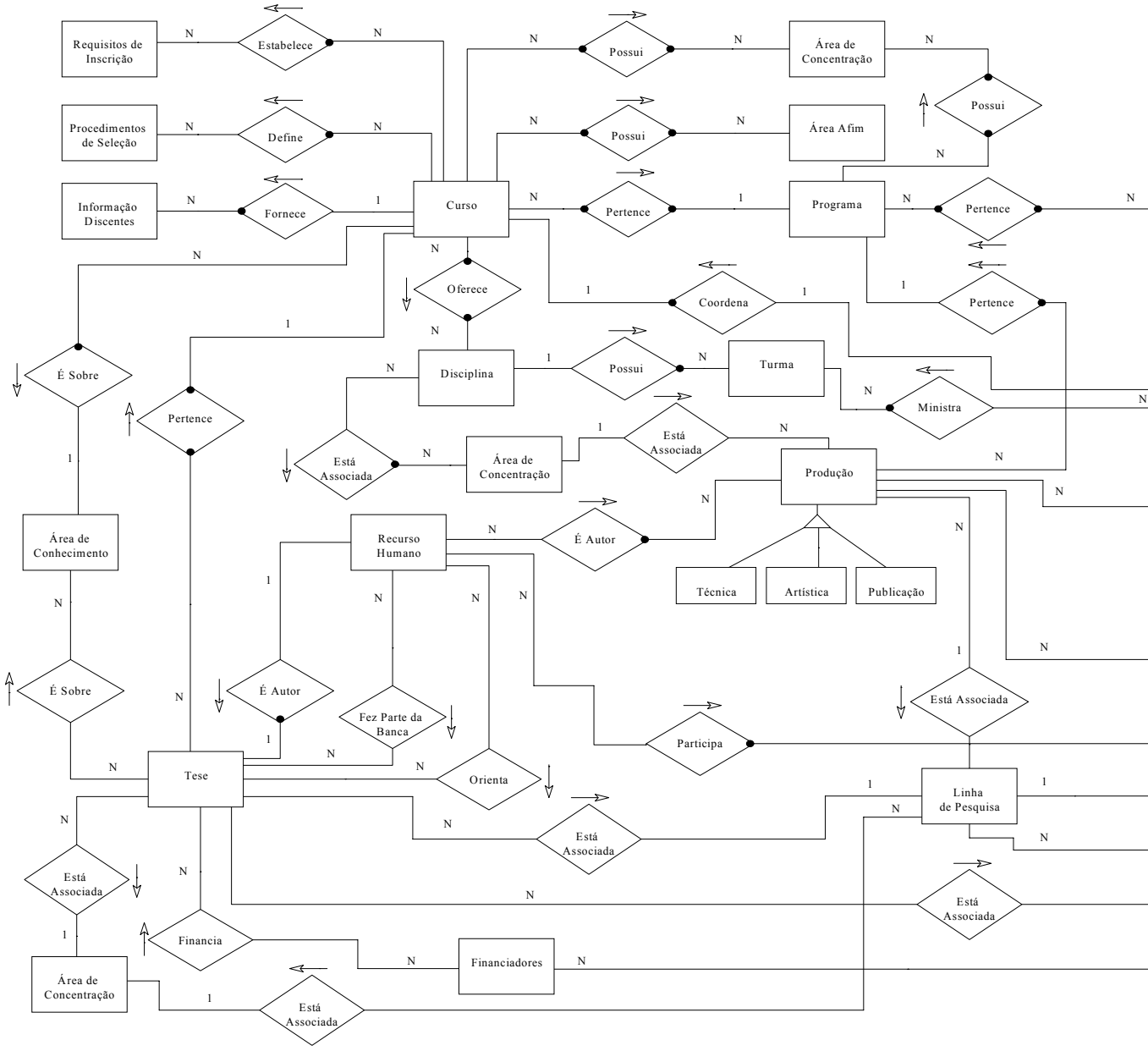


Figura 6.1: Modelo de Dados da Aplicação Coleta de Dados 5.0 (C

Atualmente, essas informações são utilizadas somente em nível operacional. Por isso, existe uma necessidade muito grande do desenvolvimento de um sistema para a descoberta de conhecimentos valiosos que possam ser utilizados no suporte à tomada de decisão. Os dados desses bancos de dados operacionais são atualizados anualmente e na abertura de um novo curso de pós-graduação recomendado pela CAPES.

6.3.2 Definição dos Objetivos do Sistema

Como pode ser notado, a CAPES é responsável pela avaliação e credenciamento da pós-graduação *strito-senso* do país. Para tal, seus instrumentos de captura de informações, utilizados por cerca de 1400 Programas, procuram mapear as atividades realizadas anualmente no âmbito de cada Programa. O resultado é a construção de ampla base de dados que permite inspecionar diversos aspectos da pós-graduação brasileira.

Neste estudo, o principal objetivo é estabelecer a construção de sistemas de descoberta de conhecimento segundo a metodologia proposta e verificar a efetividade de seus resultados. Para tal, inspecionar-se-ão os seguintes objetivos de investigação:

1) Estudo da relação Fomento x Nível de Formação do Quadro Funcional do Programa: verificar se existe alguma relação entre quantidade de bolsas fornecidas ao Programa, sua produtividade e a titulação de seus docentes, pesquisadores, e discentes autores;

2) Estudo da relação entre a carga de orientação e tempo de titulação dos orientandos: verificar se existe alguma relação entre o tempo de formação do aluno e o total de orientandos do seu orientador;

3) Estudo da relação entre o tempo de titulação com a disponibilidade do Fomento (bolsas) no Programa ou com a participação discente em projetos: verificar se existe alguma relação entre o tempo de formação do aluno com o fato do mesmo possuir ou não bolsa e com a vinculação de sua dissertação a projetos.

O conhecimento descoberto poderá ser aplicado pela CAPES visando:

- A inspeção sobre os dados para verificar se indicadores de qualidade da pós-graduação apresentam coerência, quando relacionados;

- A verificação de hipóteses de "senso comum" comentadas na comunidade científica (ex: mais orientandos → maior tempo médio de titulação).

A tarefa de mineração de dados identificada é associação.

6.3.3 Construção do Diagrama de Classes do Sistema em Operação

Para o desenvolvimento do sistema de descoberta de conhecimento são tomadas como base algumas das tabelas dos bancos de dados da CAPES que contêm dados sobre os Programas de pós-graduação do Brasil. Essas tabelas e seus relacionamentos são representados em um único diagrama de classes, considerando que os bancos de dados utilizados pertencem somente a um único sistema, como mostra a Figura 6.2.

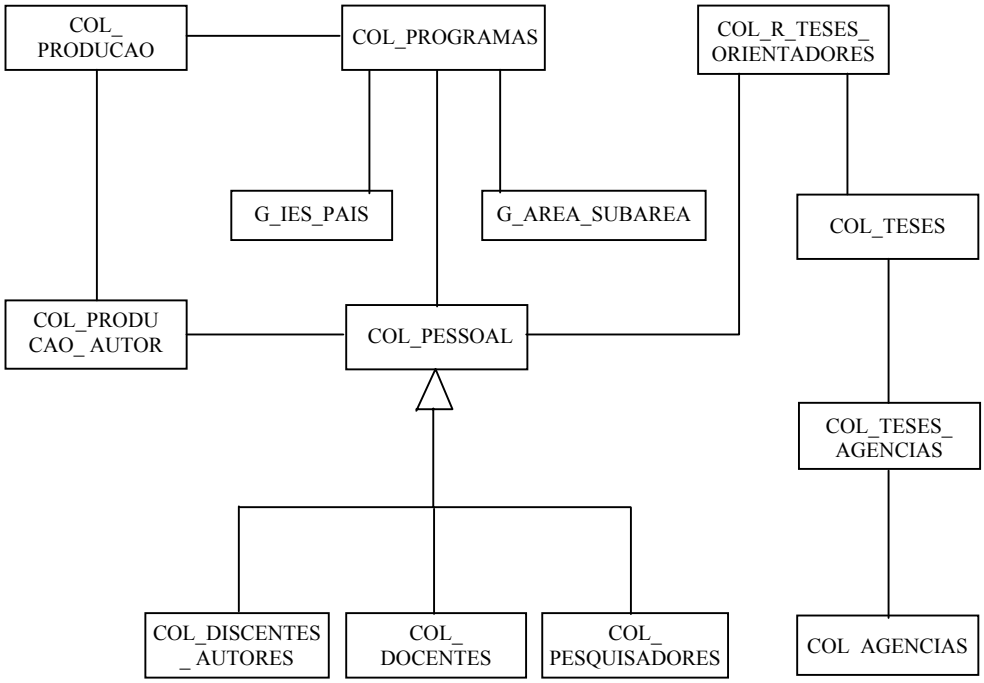


Figura 6.2: Diagrama de Classes do Sistema em Operação

6.3.4 Construção do Diagrama de Classes Corporativo

Considerando que existe apenas um diagrama de classes que representa o sistema em operação da CAPES, não há razão para construir o diagrama de classes corporativo, pois este seria idêntico ao apresentado na Figura 6.2.

6.4 PROJETO INFORMAL

A etapa de Projeto Informal está dividida em onze sub-etapas, as quais são descritas a seguir, são elas:

1) Seleção dos atributos;
2) Definição das transformações dos dados;
3) Projeto de uma estrutura de metadados;
4) Projeto do DW/DM/DS;
5) Definição de técnica de amostragem estatística;
6) Escolha de técnica de mineração de dados;
7) Construção dos diagramas de classes;
8) Construção dos diagramas de estados;
9) Construção de diagramas de colaboração;
10) Determinação da forma de povoamento do DW/DM/DS;

6.4.1 Seleção dos Atributos

A seleção dos atributos necessários é feita baseando-se nos objetivos definidos anteriormente. A Tabela 6.1 relaciona os atributos selecionados de cada tabela de banco de dados utilizada como base para o estudo de casos. A seleção de alguns atributos de identificação é necessária, neste momento, para possibilitar a junção das tabelas na fase de transformação dos dados.

Tabela 6.1: Atributos Selecionados das Tabelas dos Bancos de Dados da CAPES

Nome da Tabela de Entrada	**Atributos**
COL_PROGRAMAS	AnoBase, Codigo, IdIESPais, RdnNivel, Nome, IdAreaBasica
COL_TESES	AnoBase, IdPrograma, Sequencial, IdPessoal, MatriculaAutor, DefesaTese, Nivel, IdProjetoPesquisa
COL_R_TESES_AGENCIAS	AnoBase, IdPrograma, IdTese, NumeroMeses
COL_R_TESES_ORIENTADORES	AnoBase, IdPrograma, IdTese, IdPessoalOrientador, Tipo
COL_DISCENTES_AUTORES	AnoBase, IdPrograma, IdPessoal, NivelTitulacao
COL_DOCENTES	AnoBase, IdPrograma, IdPessoal, NumeroOrientandosMestrado, NumeroOrientandosDoutorado, TitulacaoAtual
COL_PESQUISADORES	AnoBase, IdPrograma, IdPessoal, TitulacaoAtual
COL_PRODUCAO	AnoBase, IdPrograma, AreaProdução, VinculoTese, IdTipoProducao, IdOrientadorTese
G_IES_PAÍS	Codigo, Nome, Sigla
G_AREA_SUBAREA	Código, IdGrandeArea, IdArea

6.4.2 Definição das Transformações dos Dados

Tomando como base os objetivos a serem atingidos com o sistema, alguns resumos devem ser obtidos. A seguir, para cada objetivo, são relacionadas tabelas usadas como fonte de entrada na obtenção de cada tabela de saída, que são as tabelas de *dataset* usadas como entrada para o algoritmo de mineração de dados. Os resumos são descritos nos atributos das tabelas de saída. É importante lembrar que os atributos das tabelas de entrada já foram selecionados e estão relacionados na tabela 6.1.

1) Estudo da relação Fomento x Nível de Formação do Quadro Funcional do Programa

   Tabelas de entrada:

   - COL_TESES
   - COL_R_TESES_AGENCIAS
   - COL_PRODUCAO
   - COL_DOCENTES
   - COL_PESQUISADORES
   - COL_DISCENTES_AUTORES
   - COL_PROGRAMAS
   - G_IES_PAIS
   - G_AREA_SUBAREA

   Tabela de saída:

   DS_BOLSAS_PRODUTIVIDADE

   - IdGrArea;
   - Área: descrição da área básica;
   - GrandeArea: descrição da grande área;
   - BolsaAluno: média do número de meses do financiamento por Programa (BolsaAluno = $\sum$ **NumeroMeses** / $\sum$ **Alunos**);
   - TotalProducao: somatória das produções por Programa.
   - Produtividade: média de produção por docente/pesquisador/discente autor (Produtividade = $\sum$ **Producao** / ($\sum$ **Docentes** + $\sum$ **Pesquisadores** + $\sum$ **DiscentesAutores**));
   - TotalGraduados: total de docentes, pesquisadores e discentes autores com titulação atual de graduação;
   - TotalMestres: total de docentes, pesquisadores e discentes autores com titulação atual de mestrado;

- TotalDoutores: total de docentes, pesquisadores e discentes autores com titulação atual de doutorado;

2) Estudo da relação entre a carga de orientação e tempo de titulação dos orientandos

Tabelas de entrada:

- COL_R_TESES_ORIENTADORES
- COL_TESES
- COL_DOCENTES
- COL_PROGRAMAS
- G_IES_PAIS

Tabela de saída:

DS_ORIENTANDOS_FORMACAO

- IdGrArea;
- Área: descrição a área básica;
- GrandeArea: descrição da grande área;
- TotalOrientandos: número orientandos mestrado + número orientandos doutorado por orientador;
- MediaMeses: média de meses de formação dos alunos por orientador;

3) Estudo da relação entre o tempo de titulação com a disponibilidade do Fomento (bolsas) no Programa ou com a participação discente em projetos

Tabelas de entrada:

- COL_TESES
- COL_R_TESES_AGENCIAS
- COL_PRODUCAO
- COL_PROGRAMAS
- G_IES_PAIS
- G_AREA_SUBAREA

Tabela de saída:

DS_FORMACAO_BOLSA_VINCULO

- IdGrArea;
- Área: descrição a área básica;
- GrandeArea: descrição da grande área;
- MesesFormação: total de meses para formação do aluno;
- Bolsa: 1 – com bolsa, 0 – sem bolsa;
- Vínculo: 1 – com vínculo a projeto, 0 – sem vínculo a projeto;

Os resumos definidos acima são obtidos através da definição e implementação de métodos da classe de objetos AgenteTransformação, definida no Capítulo 5. Esses métodos (relacionados na Tabela 6.2) representam, na verdade, o detalhamento do método TransformarDados.

Tabela 6.2: Relação de Atributos e Métodos do Agente Transformação

Atributos	Métodos
AnoBase, IdPrograma, Programa, Sigla, IdAreaBasica, Área, GrandeArea, IdTese, MatriculaAutor, DefesaTese, Nivel, IdPessoalOrientador, NumeroMeses, IdProjetoPesquisa, IdPessoal, TitulacaoAtual, NumeroOrientandosMestrado, NumeroOrientandosDoutorado, BolsaAluno, Produtividade, TotalGraduados, TotalMestres, TotalDoutores, TotalNotorioSaber, TotalOrientandos, MediaMeses, MesesFormacao, Bolsa, Vinculo	CalcularBolsaAluno, CalcularProducao, CalcularTotalGraduados, CalcularTotalMestres, CalcularTotalDoutores, CalcularTotalOrientandos, CalcularMediaMeses, CalcularMesesFormacao, VerificarBolsaVinculo DiscretizarDados

O método DiscretizarDados realiza as discretizações necessárias sobre os dados gerados na aplicação dos demais métodos do Agente Transformação.

Os dados de cada tabela do DataSet foram discretizados para atender a exigências do algoritmo de mineração de dados utilizado (ver seção 6.4.6). A discretização foi realizada de acordo com as especificações apresentadas na Tabela 6.3. Cada variável de entrada é dividida em faixas de valores. Cada faixa de valor é preenchida com 1 se o valor da variável de entrada pertence àquela faixa, caso contrário, ela é preenchida com 0. As faixas foram manualmente definidas após uma análise realizada sobre os valores dos dados dos cursos de pós-graduação da Universidade Federal de Santa Catarina.

6.4.3 Projeto de uma Estrutura de Metadados

Para simplificar a realização do estudo de caso proposto, foram definidas apenas duas tabelas de metadados para o registro das informações necessárias para a criação das tabelas do *data set*. Essas tabelas e seus atributos estão relacionados na Tabela 6.4. Os métodos das classes de objetos dos metadados são: Criar, Remover, GravarDefinicoes, LerMetadados e Atualizar, conforme já documentados no Capítulo 5.

Tabela 6.3: Discretização dos Dados das Tabelas do DataSet

Variável	Faixas	Valores
IdGrandeArea	Ciências Exatas e da Terra	10000003
	Ciências Biológicas	20000006
	Engenharias	30000009
	Ciências da Saúde	40000001
	Ciências Agrárias	50000004
	Ciências Sociais Aplicadas	60000007
	Ciências Humanas	70000000
	Lingüística, Letras e Artes	80000002
	Grande Área não Informada	90000000
	Outros	90000005
BolsaAluno	BolsaAluno0_12	maior ou igual a 0 e menor ou igual a 12
	BolsaAluno13_18	maior ou igual a 13 e menor ou igual a 18
	BolsaAluno19_24	maior ou igual a 19 e menor ou igual a 24
	BolsaAluno25_30	maior ou igual a 25 e menor ou igual a 30
	BolsaAluno31_	maior ou igual a 31
Produtividade (Quantitativa)	Produtiv0_1	maior ou igual a 0 e menor ou igual a 1
	Produtiv2_3	maior ou igual a 2 e menor ou igual a 3
	Produtiv4_6	maior ou igual a 4 e menor ou igual a 6
	Produtiv7_10	maior ou igual a 7 e menor ou igual a 10
	Produtiv11_	maior ou igual a 11
TotalProducao	TotProd0_50	maior ou igual a 0 e menor ou igual a 50
	TotProd50_100	maior ou igual a 51 e menor ou igual a 100
	TotProd101_200	maior ou igual a 101 e menor ou igual a 200
	TotProd201_300	maior ou igual a 201 e menor ou igual a 300
	TotProd301_400	maior ou igual a 301 e menor ou igual a 400
	TotProd401_	maior ou igual a 401
TotalGraduados	TotalGrad0_5	maior ou igual a 0 e menor ou igual a 5
	TotalGrad6_20	maior ou igual a 6 e menor ou igual a 20
	TotalGrad21_40	maior ou igual a 21 e menor ou igual a 40
	TotalGrad41_70	maior ou igual a 41 e menor ou igual a 70
	TotalGrad71_100	maior ou igual a 71 e menor ou igual a 100
	TotalGrad101	maior ou igual a 101
TotalMestres	TotalMest0_20	maior ou igual a 0 e menor ou igual a 20
	TotalMest21_40	maior ou igual a 21 e menor ou igual a 40
	TotalMest41_20	maior ou igual a 41 e menor ou igual a 70
	TotalMest71_20	maior ou igual a 71 e menor ou igual a 100
	TotalMest101_200	maior ou igual a 101 e menor ou igual a 200
	TotalMest201_	maior ou igual a 201
TotalDoutores	TotalDout0_20	maior ou igual a 0 e menor ou igual a 20
	TotalDout21_40	maior ou igual a 21 e menor ou igual a 40
	TotalDout41_60	maior ou igual a 41 e menor ou igual a 60
	TotalDout61_80	maior ou igual a 61 e menor ou igual a 80
	TotalDout81_100	maior ou igual a 81 e menor ou igual a 100
	TotalDout101_	maior ou igual a 101
NumeroOrientandos	NumOrient_1	igual a 1
	NumOrient_2	igual a 2
	NumOrient_3	igual a 3
	NumOrient4_7	maior ou igual a 4 e menor ou igual a 7
	NumOrient8_	maior ou igual a 8
MesesFormacao	MesesForm0_12	maior ou igual a 0 e menor ou igual a 12
	MesesForm13_18	maior ou igual a 13 e menor ou igual a 18
	MesesForm19_24	maior ou igual a 19 e menor ou igual a 24
	MesesForm25_30	maior ou igual a 25 e menor ou igual a 30
	MesesForm31_36	maior ou igual a 31 e menor ou igual a 36
	MesesForm36_	maior ou igual a 36

Tabela 6.4: Relação das Tabelas do Metadados e seus Atributos

Nome da tabela	Atributos
MD_METADADO	id, nome, sqlCriacao, sqlExtracao, Descrição
MD_CATEGORIA	idTabela, id, Origem, Mínimo, Maximo, Destino

6.4.4 Projeto do *Data Set*

Para o projeto do *data set*, são relacionadas as tabelas e seus atributos, conforme mostra a Tabela 6.5. Após ser escolhida a técnica de mineração de dados, novos atributos poderão ser definidos. Os métodos da classe de objetos DataSet são os mesmos para todas as tabelas definidas, são eles: Criar, Remover, GravarDataSet e LerDataSet.

Tabela 6.5: Relação das Tabelas do DataSet e seus Atributos

Nome da tabela	Atributos	Unidade Básica de Análise
DS_BOLSAS_PRODUTIVIDADE	IdGrArea, Área, GrandeArea, BolsaAluno, TotalProducao, Produtividade, TotalGraduados, TotalMestres, TotalDoutores	Programa de Pós-Graduação
DS_ORIENTANDOS_FORMACAO	IdGrArea, Área, GrandeArea, TotalOrientandos, MediaMeses	Orientador
DS_FORMACAO_BOLSA_VINCULO	IdGrArea, Área, GrandeArea, MesesFormacao, Bolsa, Vinculo	Orientando

O número de tuplas obtidas em cada tabela do *dataset* foi:

- DS_BOLSAS_PRODUTIVIDADE: 474;
- DS_ORIENTANDOS_FORMACAO: 4981;
- DS_FORMACAO_BOLSA_VINCULO: 8398.

6.4.5 Definição de Técnica de Amostragem Estatística

Por ser relativamente pequeno o volume de dados de entrada, não é necessário definir uma amostra dos dados. Foram definidos apenas resumos e discretizações durante a fase de definição das transformações dos dados.

6.4.6 Escolha de Técnica de Mineração de Dados

Para o estudo de caso em questão, foi aplicada a técnica de mineração de dados "Descoberta de Regras de Associação", mais especificamente o algoritmo "apriori", implementado na linguagem Delphi 5.0 por Gonçalves (2000).

Essa escolha foi baseada nos objetivos do sistema que, de forma geral, são descobrir associações entre os dados e na disponibilidade de um componente de *software* que implementa o algoritmo "apriori".

Esse algoritmo tem como parâmetros de entrada os dados dos Programas de Pós-Graduação do Brasil, já preparados e armazenados no *data set*, o suporte mínimo e o grau de confiança. Os dois últimos parâmetros podem ser escolhidos e ajustados na busca de resultados que levem aos objetivos relacionados na etapa de análise do sistema. Os valores mínimos de suporte e confiança utilizados nos experimentos foram 8% e 40%, respectivamente.

6.4.7 Construção de Diagramas de Classes

Os diagramas de classes, necessários para o sistema de descoberta de conhecimento sendo especificado, são de dois tipos: o primeiro representa os bancos de dados do sistema em operação (Figura 6.3) e o segundo representa o *data set* definido para o estudo de caso (Figura 6.4). Nesses diagramas de classes não estão relacionados os métodos por serem os mesmos para as classes de objetos que representam as tabelas dos bancos de dados do sistema em operação (Criar, Remover, VerAtributos e LerBancoDados) e, também os mesmos para as classes de objetos que representam as tabelas do *data set* (Criar, Remover, GravarDataSet e LerDataSet).

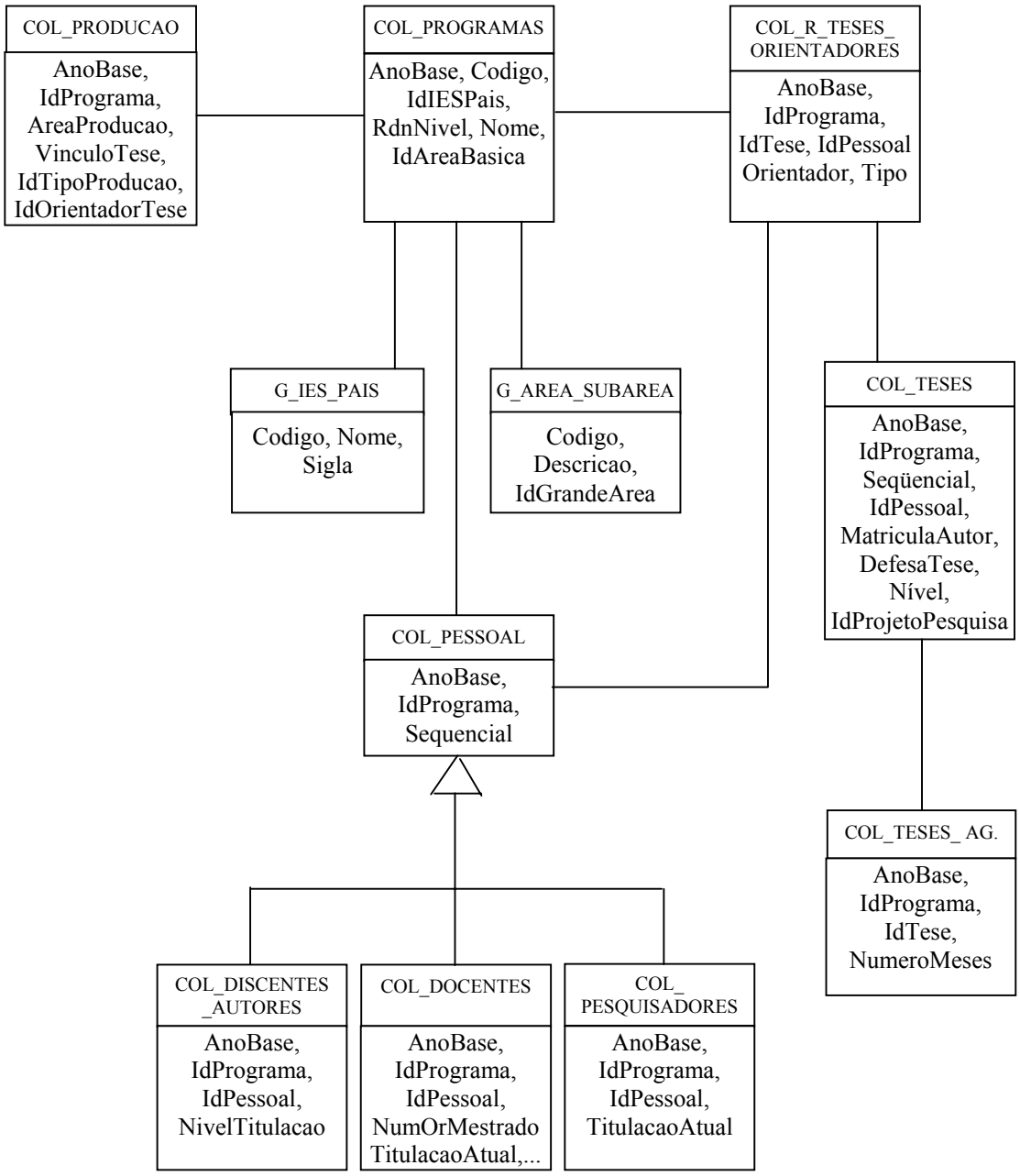


Figura 6.3: Diagrama de Classes Refinado dos Bancos de Dados em Operação

Figura 6.4: Diagrama de Classes do *DataSet*

6.4.8 Construção de Diagramas de Estados

A Figura 6.5 mostra o diagrama de estados da classe de objetos AgenteTransformacao, mais especificamente, o detalhamento do método TransformarDados apresentado na Figura 9.13 (Anexo 9.2).

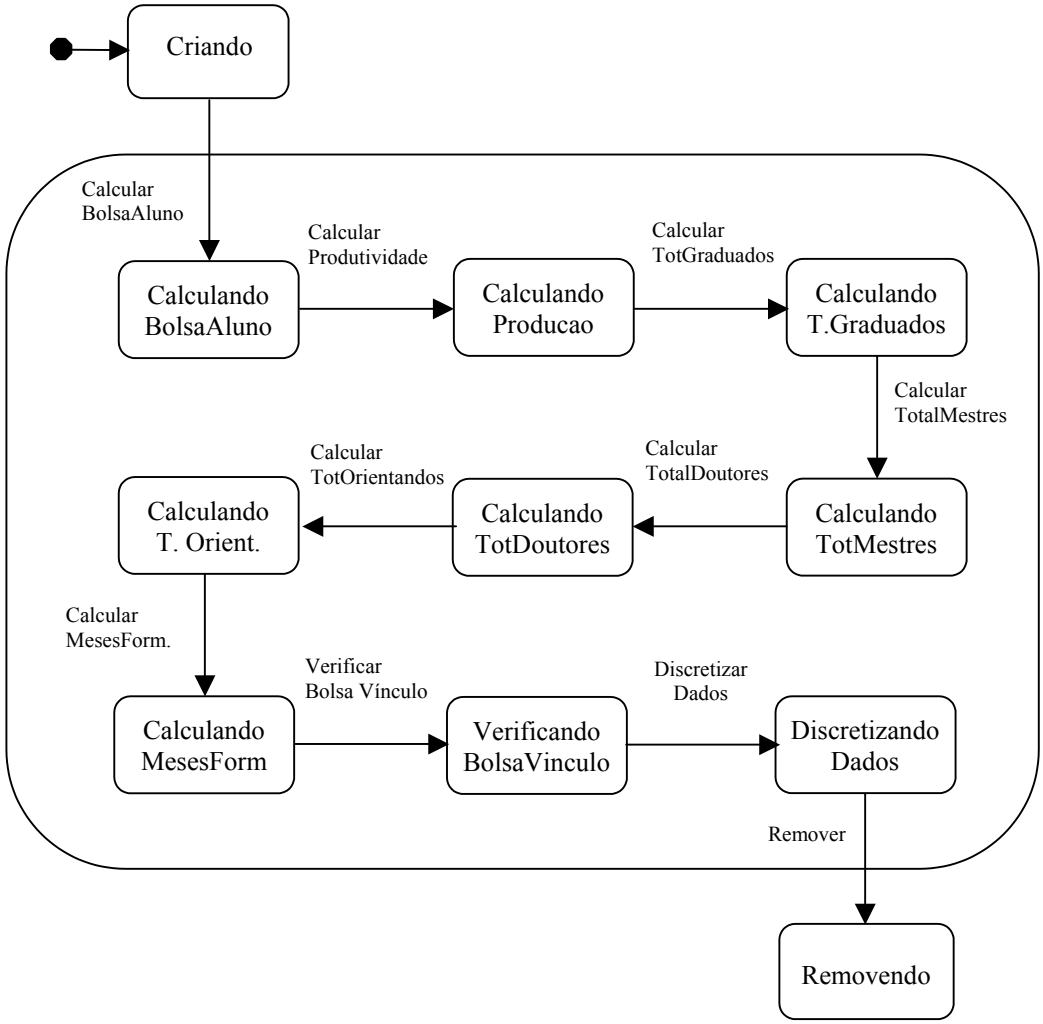


Figura 6.5: Diagrama de Estados da Classe AgenteTransformação

Para as outras classes de objetos que representam as tabelas dos bancos de dados de entrada (do sistema em operação), do *data set* e do metadados, não são construídos diagramas de estados pelo fato desses diagramas já terem sido construídos durante a especificação do ambiente ADesC (ver Figuras 9.19, 9.20 e 9.21 do Anexo 9.2).

6.4.9 Construção de Diagramas de Colaboração

A seguir, as Figuras 6.6 a 6.13 representam os diagramas de colaboração dos métodos que implementam a transformação de dados realizada pelo Agente de Transformação.

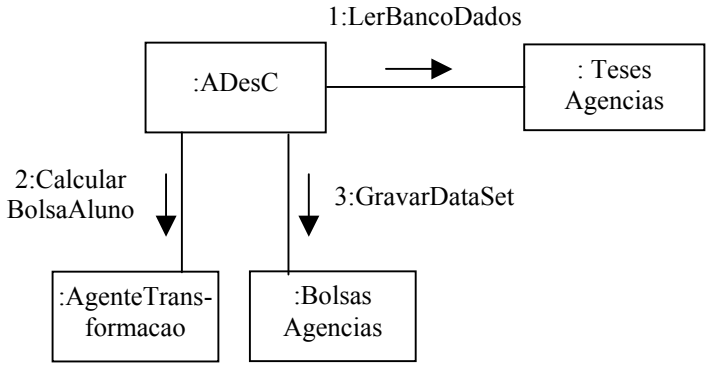


Figura 6.6: Diagrama de Colaboração – CalcularBolsaAluno

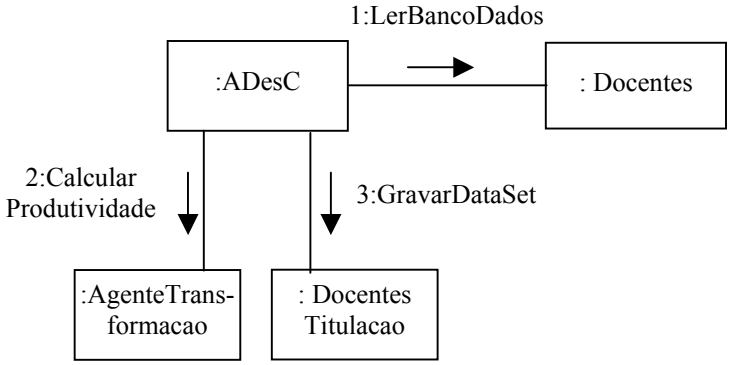


Figura 6.7: Diagrama de Colaboração – CalcularProducao

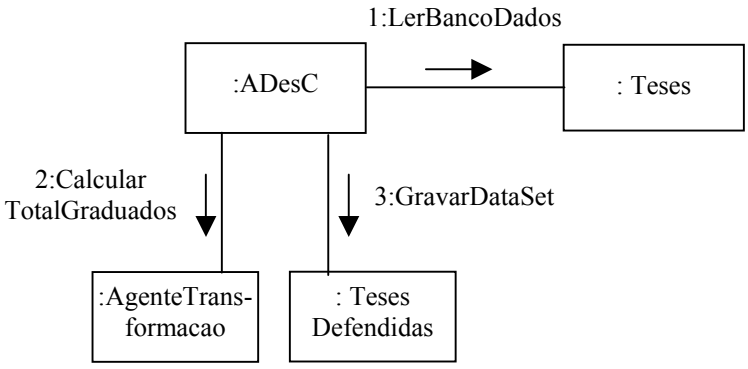


Figura 6.8: Diagrama de Colaboração – CalcularTotalGraduados

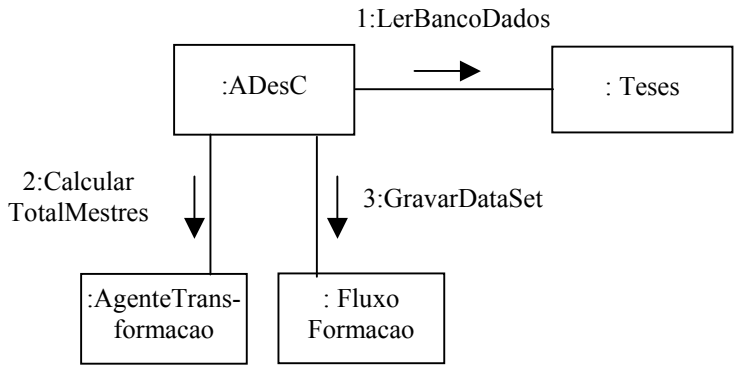


Figura 6.9: Diagrama de Colaboração – CalcularTotalMestres

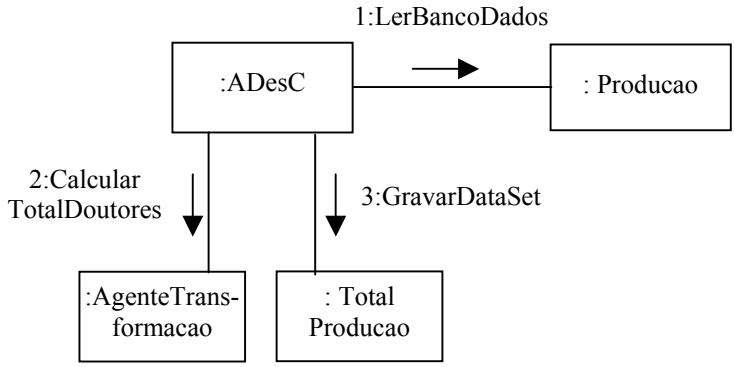


Figura 6.10: Diagrama de Colaboração – CalcularTotalDoutores

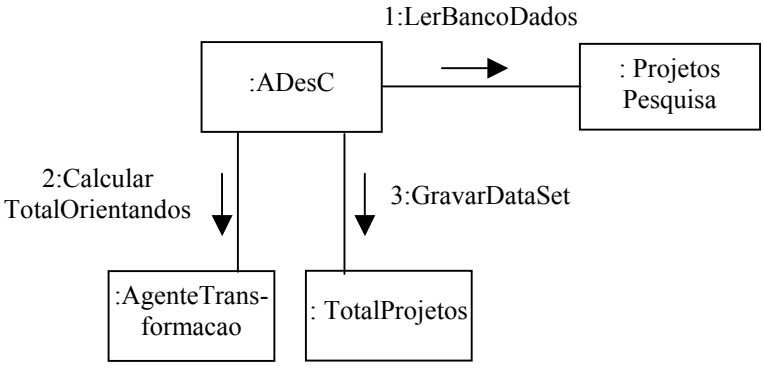


Figura 6.11: Diagrama de Colaboração – CalcularTotalOrientandos

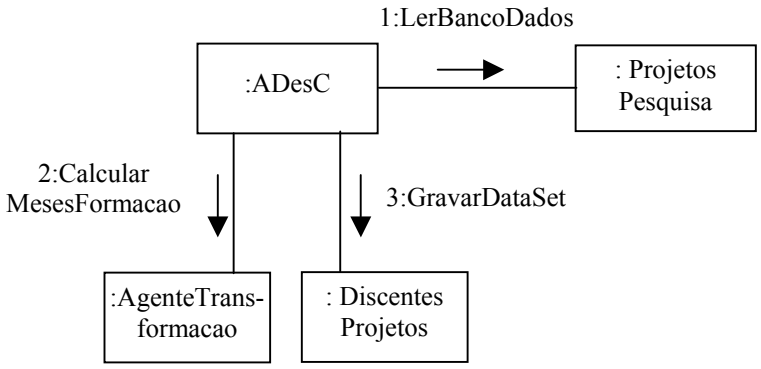


Figura 6.12: Diagrama de Colaboração – CalcularMesesFormacao

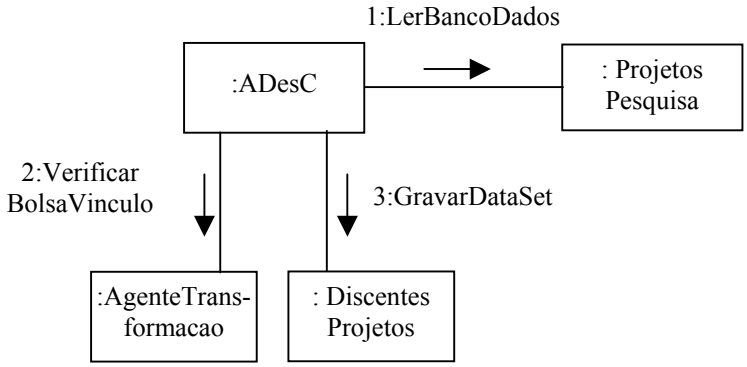


Figura 6.13: Diagrama de Colaboração – VerificarBolsaVinculo

6.4.10 Determinação da Forma de Povoamento do *Data Set*

O povoamento do *data set* pode ser realizado de acordo com os seguintes eventos:

- Sempre que a CAPES receber dados sobre a criação de um novo curso de pós-graduação.
- Trimestralmente para que sejam atualizados os dados dos cursos de pós-graduação já existentes.
- No momento em que for realizada uma investigação a respeito dos dados da CAPES, ou seja, antes de ser aplicada uma técnica de mineração de dados.

6.5 PROJETO FORMAL

Como já foi visto no Capítulo 4 desta tese, a construção do modelo formal em E-LOTOS é realizada através dos seguintes passos:

1) Divisão do sistema em módulos;
2) Definição do modelo de comunicação;
3) Declaração dos módulos.

6.5.1 Divisão do Sistema em Módulos

A divisão do sistema em módulos foi realizada no projeto informal, considerando que cada classe de objetos é representada como um módulo.

6.5.2 Definição do Modelo de Comunicação

O modelo de comunicação utilizado neste passo de formalização é o mesmo definido no Capítulo 5, representado na Figura 5.15.

6.5.3 Declaração dos Módulos

A especificação formal das classes de objetos que representam as tabelas dos bancos de dados do sistema em operação, do *data set* e do metadados pode ser encontrada no Anexo 9.4.

A seguir, são especificados em E-LOTOS os métodos que implementam a rotina de transformação de dados do módulo *modAgTransformação*. Portanto, a especificação desses métodos deve substituir o processo *TransformarDados* mostrada no Anexo 9.4.

```
module modAgTransformacao import modDados is
...
  function CalcularMesesFormacao (periodo: Typetec) : integer is
        var totalmeses: integer
        in
        (* Calcular o total de meses gastos para a realização do curso por aluno*)
            hide ent_meses: integer in
                write (“Entrar o total de meses no curso por aluno”) ;
                ent_meses (?totalmeses) ;
            endhide
        endvar
  endfunc
  process AgenteTransf [peat: PortaEntAT, psat: PortaSaiAT] is
        TransformarDados [peat, psat]
```

```
endproc
process TransformarDados [peat: PortaEntAT, psat: PortaSaiAT] is
        CalcularBolsaAluno [peat, psat]
    []  CalcularTotalProducao [peat, psat]
    []  CalcularTotalGraduados [peat, psat]
    []  CalcularTotalMestres [peat, psat]
    []  CalcularTotalDoutores [peat, psat]
    []  CalcularTotalOrientandos [peat, psat]
    []  CalcularMediaMeses [peat, psat]
    []  VerificarBolsaVinculo [peat, psat]
    []  DiscretizarDados[peat, psat]
endproc
process CalcularBolsaAluno [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular média do número de meses de financiamento concedido aos alunos *)
            hide ent_dset: Typebd in
                write (“Entrar com a média de meses de financiamento”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
endproc
process CalcularTotalProducao [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular o total de produção intelectual por Programa *)
            hide ent_dset: Typebd in
                write (“Entrar com o total de produção intelectual por área e por Programa”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
endproc
process CalcularTotalGraduados [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular total de graduados docentes, pesquisadores e discentes autores por
Programa *)
```

```
            hide ent_dset: Typebd in
                write (“Entrar com o total de graduados por Programa”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
  process CalcularTotalMestres [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular o total de mestres docentes, pesquisadores e discentes autores por
Programa *)
            hide ent_dset: Typebd in
                write (“Entrar com o total de mestres por Programa”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
  process CalcularTotalDoutores [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular total de doutores docentes, pesquisadores e discentes autores por
Programa *)
            hide ent_dset: Typebd in
                write (“Entrar com o total de doutores por Programa”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
  process CalcularTotalOrientandos [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Calcular total de orientandos por orientador e por Programa *)
            hide ent_dset: Typebd in
                write (“Entrar com o total de orientandos por orientador e por Programa”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
```

```
            endhide
        endvar
  endproc
  process CalcularMediaMeses [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
            ?totalmeses :=CalcularMesesFormacao (?inftransf);
         (* Calcular a média de tempo gasto para formação dos alunos por Programa*)
            hide ent_dset: Typebd in
                write ("Entrar com a média de meses de formação por Programa") ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
  process VerificarBolsaVinculo [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Identificar se o aluno tem bolsa e se a sua tese tem algum vinculo com projetos *)
            hide ent_dset: Typebd in
                write ("Entrar com '0' ou '1' para Bolsa e para Vinculo por aluno") ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
  process DiscretizarDados [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Realizar a discretização dos dados *)
            hide ent_dset: Typebd in
                write ("Entrar com o nome de cada dataset ") ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
  endproc
endmod (* AgenteTransf *)
```

6.6 IMPLEMENTAÇÃO DO SISTEMA

A implementação foi feita através do protótipo do ambiente ADesC, especificado no Capítulo 5 desta tese.

Os dados utilizados foram dos cursos de pós-graduação das seguintes instituições de ensino do Brasil, por região do país[1]:

- **Região Sul**: UFSC, UFRGS, UFPR, CEFET/PR, PUC/PR, PUC/RS, UEL, UFPEL, FURG, UNISINOS, FFFCMPA, UPF.
- **Região Sudeste**: UFRJ, UFU, UFMG, UFSCAR, UNICAMP, USP, USP/CENA, USP/RP, UMESP, UNESP, UNESP/ARAR, UNESP/ASS, UNESP/BOT, UNESP/GUAR, UNESP/JAB, UNESP/MAR, UNESP/RC, UNESP/SJC, UNESP/SJRP, UNI-RIO, UNIMEP, UGF, UNIFESP.
- **Região Nordeste**: UFBA, UFPB/CG, UFPB/JP, UFPE, UFRN, UFC, UFPA.
- **Região Centro-Oeste**: UFG, UFMT, UNB.

6.7 ANÁLISE DOS RESULTADOS

A análise dos resultados baseou-se na abordagem objetiva, ou seja, procurou-se selecionar as regras de associação de maior interesse conforme os valores de suporte e confiança e em consistência com os dados. De acordo com esses objetivos, tem-se:

1) Estudo da relação Fomento x Nível de Formação do Quadro Funcional do Programa
   A Tabela 6.6 relaciona as regras geradas para este estudo.

[1] Não foi possível utilizar os bancos de dados operacionais de todos os cursos de pós-graduação do Brasil porque houve problema na descompressão de alguns deles.

Tabela 6.6: Regras Geradas para o Primeiro Objetivo

Regra	Suporte	Confiança
Se o Programa possui de 21 a 40 mestres, então ele produz de 2 a 3 publicações em média por pesquisador por ano	21,73%	45,18%
Se o Programa possui de 19 a 24 meses de financiamento por aluno concluído, então ele produz de 2 a 3 publicações em média por pesquisador	19,62%	53,76%
Se o Programa possui de 25 a 30 meses de financiamento por aluno concluído, então ele produz de 2 a 3 publicações por pesquisador	16,88%	48,19%
Se o Programa possui de 25 a 30 meses de financiamento por aluno concluído, então ele possui de 21 a 40 Mestres	15,19%	43,37%

Interpretação das Regras:

- **21 a 40 mestres implica em Produtividade 2 a 3**

  21,73% dos Programas de pós-graduação do Brasil possuem de 21 a 40 mestres no corpo docente e discente e produzem de 2 a 3 publicações em média por pesquisador[2].

  45,18% dos Programas de pós-graduação do Brasil que possuem de 21 a 40 mestres produzem de 2 a 3 publicações em média por pesquisador.

- **Bolsa x Aluno 19 a 24 meses implica em Produtividade 2 a 3**

  19,62% dos Programas possuem em média 19 a 24 meses de financiamento por aluno concluído e produzem de 2 a 3 publicações em média por pesquisador.

  53,76% dos Programas que possuem em média 19 a 24 meses de financiamento por aluno concluído produzem de 2 a 3 publicações em média por pesquisador.

- **Bolsa x Aluno 25 a 30 meses implica em Produtividade 2 a 3**

  16,88% dos Programas possuem em média 25 a 30 meses de financiamento por aluno e produzem de 2 a 3 publicações por pesquisador.

  48,19% dos Programas que possuem em média 25 a 30 meses de financiamento por aluno concluído produzem de 2 a 3 publicações por pesquisador.

[2] Pesquisador neste contexto inclui, também, docente e discente-autor.

- **Bolsa x Aluno 25 a 30 meses implica em 21 a 40 Mestres**
  15,19% dos Programas possuem em média 25 a 30 meses de financiamento por aluno concluído e possuem de 21 a 40 mestres.
  43,37% dos Programas que possuem em média 25 a 30 meses de financiamento por aluno concluído possuem 21 a 40 mestres.

Pode-se observar que o aumento relativo na quantidade de bolsas de um Programa implica em aumento na Produção, mas há um nível de saturação para esta regra (quando média de bolsa por aluno está entre 25 e 30 meses). É interessante observar que esta constatação vai de encontro à política de redução do tempo máximo de bolsa adotada pela CAPES.

As regras referentes à Titulação do corpo Docente e Discente não permitem o mesmo tipo de análise, dado que apresentaram relações absolutas e independentes.

2) Estudo da relação entre a carga de orientação e tempo de titulação dos orientandos:
   A Tabela 6.7 relaciona as regras geradas para este estudo.

Tabela 6.7: Regras Geradas para o Segundo Objetivo

Regra	Suporte	Confiança
Se o orientador possui de 4 a 7 orientandos, então a média de tempo de formação de seus orientandos de mestrado é maior que 36 meses	16,92%	49,21%
Se o orientador possui 1 orientando, então o tempo de formação de seu orientando de mestrado é maior que 36 meses	8,43%	41,71%

Interpretação das Regras:

- **Número Orientandos 4 a 7 implica em Meses Formação > 36**
  16,92% dos orientadores têm de 4 a 7 orientandos e o tempo médio de formação de seus orientandos é maior que 36 meses.
  49,21% dos orientadores que têm de 4 a 7 orientandos, o tempo médio de formação de seus orientandos é maior que 36 meses.
- **Número Orientandos 1 implica em Meses Formação > 36**
  8,43% dos orientadores têm 1 orientando e o tempo de formação de seu orientando é maior que 36 meses.

41,71% dos orientadores que têm 1 orientando, o tempo de formação de seu orientando é maior que 36 meses.

O estudo mostra que há um equilíbrio nos valores de confiança entre o número de orientandos de um orientador e o tempo médio de formação de seus orientandos. Isto indica que não há suporte para a afirmação que alunos de um professor com mais orientandos tenham maior tempo médio de titulação.

3) Estudo da relação entre o tempo de titulação com a disponibilidade do Fomento (bolsas) no Programa ou com a participação discente em projetos:
A Tabela 6.8 relaciona as regras geradas para este estudo.

Tabela 6.8 Regras Geradas para o Terceiro Objetivo

Regra	Suporte	Confiança
Se o aluno possui bolsa, então sua dissertação está vinculada a projeto de pesquisa	36,88%	53,27%
Se o aluno possui bolsa, então seu tempo de formação é maior que 36 meses	28,34%	40,94%
Se a dissertação está vinculada a projeto, então o tempo de formação do aluno é maior que 36 meses	21,30%	42,72%
Se o aluno possui bolsa e sua dissertação está vinculada a projeto, então seu tempo de formação é maior que 36 meses	14,11%	38,26%
Se a área é Ciências Exatas e da Terra, então o aluno possui bolsa	10,16%	82,66%
Se a área é Ciências Humanas, então o aluno possui bolsa	13,06%	72,70%
Se a área é Engenharia, então o aluno possui bolsa	11,44%	65,78%
Se a área é Engenharia, então a dissertação possui vínculo com projeto	10,04%	57,70%

Interpretação das Regras:

- **Bolsa implica em Vínculo com projeto**

  36,88% dos alunos possuem bolsa e sua dissertação tem vínculo com projeto.

  53,27% dos alunos que possuem bolsa, sua dissertação tem vínculo com projeto.

- **Bolsa implica em meses de formação maior que 36**

  28,34% dos alunos possuem bolsa e seu tempo de formação é maior que 36 meses.

  40,94% dos alunos que possuem bolsa, seu tempo de formação é maior que 36 meses.

- **Vínculo com projeto implica em meses de formação maior que 36**

  21,30% das dissertações têm vínculo com projeto e o tempo de formação do aluno é maior que 36 meses.

  42,72% das dissertações que têm vínculo com projeto, o tempo de formação do aluno é maior que 36 meses.

- **Bolsa e vínculo com projeto implica em meses de formação maior que 36**

  14,11% dos alunos de mestrado possuem bolsa e vínculo com projeto e o tempo de formação do aluno é maior que 36 meses.

  38,26% dos alunos de mestrado que possuem bolsa e vínculo com projeto, o tempo de formação do aluno é maior que 36 meses.

- **Ciências Exatas e da Terra implica em bolsa**

  10,16% dos alunos de mestrado são da área de Ciências Exatas e da Terra e possuem bolsa.

  82,66% dos alunos de mestrado da área de Ciências Exatas e da Terra possuem bolsa.

- **Ciências Humanas implica em bolsa**

  13,06% dos alunos de mestrado são da área de Ciências Humanas e possuem bolsa.

  72,70% dos alunos de mestrado da área de Ciências Humanas possuem bolsa.

- **Engenharia implica em bolsa**

  11,44% dos alunos de mestrado são da área de Engenharia e possuem bolsa;

  65,78% dos alunos de mestrado da área de Engenharia possuem bolsa.

- **Engenharia implica em vínculo com projeto**

  10,04% dos alunos de mestrado são da área de Engenharia e possuem vínculo com projeto.

  57,70% dos alunos de mestrado da área de Engenharia possuem vínculo com projeto.

O estudo permitiu constatar que há um aumento no tempo médio de titulação quando o aluno possui bolsa ou quando sua dissertação está relacionada a projeto de pesquisa.

O estudo também comparou a implicação de participação de bolsistas em projetos. A regra permite afirmar que o impacto da concessão de bolsa na vinculação com projeto é maior do que no tempo médio de titulação.

Constatou-se ainda que os Programas das áreas de Ciências Exatas e da Terra, Ciências Humanas e Engenharia são aqueles que possuem o maior número de bolsas e, também, que mais da metade das dissertações da área de Engenharia possui vínculo com projeto.

6.8 CONSIDERAÇÕES FINAIS

No estudo de casos realizado pode-se constatar que a metodologia proposta, definida no Capítulo 4, pode ser utilizada por completo no desenvolvimento de sistemas de descoberta de conhecimento em banco de dados.

As etapas e sub-etapas definidas na metodologia são adequadas e ajudam muito no processo de desenvolvimento desses sistemas, por apresentarem uma forma sistemática e rigorosa a ser seguida e por utilizarem modelos gráficos na representação das características do sistema.

Os diagramas UML, usados para representar os componentes das etapas da metodologia, são de suma importância na compreensão do sistema em operação e na especificação do novo sistema.

Com a especificação do ambiente ADesC (Capítulo 5) e sua implementação, o trabalho do analista de sistemas de descoberta de conhecimento em banco de dados resume-se, basicamente, na análise do(s) sistema(s) em operação, na definição dos objetivos do sistema de descoberta de conhecimento em banco de dados, na seleção de tabelas e atributos necessários do(s) banco(s) de dados operacional(is) ou *data warehouse*, na definição das transformações dos dados, na entrada dos parâmetros solicitados pelo ambiente e na análise dos resultados obtidos pelo ambiente.

A utilização de uma linguagem de especificação formal, como E-LOTOS, impõe mais rigor no processo de desenvolvimento de sistemas, ajuda no entendimento do sistema e, principalmente, na verificação e validação da especificação do mesmo.

O desenvolvimento de sistemas de descoberta de conhecimento utilizando a metodologia MeDesC e o ambiente ADesC fornece meios para a obtenção de sistemas mais confiáveis e de melhor qualidade. Nesta tese, a análise destas características fundamenta-se na prototipação de casos reais, a exemplo do que é indicado por metodologias de desenvolvimento de sistemas com a inclusão de formalismo (seção 2.3.5 do Capítulo 2).

A etapa de análise do sistema requer do analista de sistemas conhecimento profundo do negócio da empresa e dos dados contidos nos bancos de dados operacionais.

A fase de preparação dos dados (seleção e transformação de dados) pode exigir um grande esforço do analista de sistemas se as fontes de dados estiverem espalhadas por vários bancos de dados.

A utilização de uma linguagem de consulta a banco de dados, como SQL, não é suficiente na fase de preparação de dados, porque as tarefas de seleção e transformação dos dados requerem mais recursos do que este tipo de linguagem oferece.

A escolha da técnica de mineração de dados a ser utilizada também não é uma tarefa fácil, considerando que diferentes tecnologias podem ser aplicadas ao tipo de problema definido e ao tipo de dados de fomento em Pós-Graduação.

Os resultados de mineração de dados obtidos no estudo de casos, apesar da confiança não chegar a 50% em muitas regras, permitiram levantar hipóteses da justificativa para políticas organizacionais (redução no tempo máximo das bolsas de pós-graduação pode ser justificável como instrumento de diminuição do tempo médio de titulação) e elucidar características relevantes de discussão no âmbito da comunidade científica e de avaliação de cursos de pós-graduação (por exemplo, o fato de não necessariamente o aumento no total de orientandos implicar em maior tempo de titulação dos orientados e do acréscimo no tempo médio de bolsas de um programa não implicar em maior adesão a projetos por parte do corpo discente).

O Capítulo 7 apresenta as conclusões e sugestões para trabalhos futuros.

7 CONCLUSÕES E TRABALHOS FUTUROS

7.1 CONCLUSÕES

A pesquisa teórica realizada mostrou que, apesar de existir um grande número de metodologias de desenvolvimento de sistemas computacionais, ainda são encontradas muitas dificuldades no desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Isto ocorre, geralmente, devido à característica de indeterminismo desses sistemas que os diferenciam muito de outros tipos de sistemas e, também, pela falta de uma metodologia específica para o desenvolvimento desses sistemas que seja completa e inclua formalismo visando a garantia de obtenção de sistemas confiáveis e de qualidade.

Por outro lado, existem ferramentas de mineração de dados que oferecem muitos recursos ao tomador de decisões. No entanto, essas ferramentas são, na maioria, limitadas à solução de problemas específicos de descoberta de conhecimento e pressupõem uma metodologia '*ad-hoc*' de desenvolvimento, ou seja, não se constituem em ambientes de construção de sistemas que contemplam desde a análise dos negócios da organização à visualização de resultados.

Por serem fundamentadas em metodologias clássicas, as iniciativas de desenvolvimento e aplicação de sistemas de descoberta de conhecimento não possuem formalização.

Para suprir essas dificuldades, este trabalho propôs um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados. Esse modelo engloba uma metodologia rigorosa e sistemática para esses sistemas, denominada MeDesC; um ambiente interativo para sua implementação, denominado ADesC; e uma ferramenta de simulação e validação de especificações de sistemas em E-LOTOS. A metodologia proposta define uma forma de mapeamento dos modelos UML para E-LOTOS. O ambiente ADesC baseia-se na tecnologia de agentes para facilitar o desempenho de suas tarefas.

Com o estudo de caso realizado, pode-se concluir que a metodologia MeDesC, seguida etapa por etapa, leva ao desenvolvimento eficiente e eficaz de sistemas de descoberta de conhecimento em banco de dados.

O estudo de casos serviu, também, para mostrar que a obtenção de sistemas confiáveis e de qualidade pode ser garantida com a inclusão de métodos formais no seu processo de desenvolvimento.

As funções básicas do ambiente ADesC foram implementadas, como um protótipo, para tornar possível o desenvolvimento completo do sistema definido como estudo de casos. No entanto, não foi possível verificar e validar a especificação formal deste ambiente por não existir uma ferramenta de simulação e validação de especificações em E-LOTOS.

A integração de modelagem orientada a objetos, métodos formais, técnicas de mineração de dados e agentes inteligentes proporcionam muitas vantagens no desenvolvimento de sistemas de descoberta de conhecimento, tais como:

a) O analista possui meios para comparar a análise de requisitos e objetivos iniciais do sistema com os resultados obtidos, segundo metodologia formal de implementação;
b) A metodologia proposta alia as vantagens da modelagem orientada a objetos (herança, reuso, encapsulamento, etc) de UML (especificação, diagramas de classes, de colaboração e de estados) ao processo de formalização de sistemas de descoberta de conhecimento;
c) O ambiente baseado em agentes permite a efetivação da metodologia, de forma integrada e completa, levando à implementação de sistemas de descoberta de conhecimento que sejam confiáveis e de qualidade.

A aplicação da metodologia e do ambiente propostos, tomando como estudo de casos os dados da pós-graduação brasileira, permitiu verificar a viabilidade e a utilidade prática do modelo proposto em um caso real. Além disso, os resultados do sistema desenvolvido podem apoiar políticas organizacionais adotadas e elucidar características relevantes de discussão no âmbito da comunidade científica e de avaliação de cursos de pós-graduação.

As conclusões do estudo de casos permitiram mostrar a relevância da metodologia MeDesC na obtenção de resultados de mineração de dados partindo-se de hipóteses levantadas por usuários e buscando-se, passo a passo, meios de se chegar à prova verdadeira ou falsa dessas hipóteses.

As dificuldades encontradas no decorrer desta pesquisa estão relacionadas à diversificação de tecnologias avançadas empregadas, no entendimento da linguagem de formalização E-LOTOS, na especificação formal do ambiente, na definição do estudo de casos e na preparação de dados do estudo de casos.

O modelo de dados relacional pode ser considerado como uma limitação do ambiente especificado. No entanto, outros modelos de dados poderão ser considerados na etapa de implementação do ambiente.

Outra limitação deste trabalho é a utilização de comandos SQL, pelo protótipo, na fase de preparação de dados.

As principais contribuições desta pesquisa são:

- Uma metodologia completa para o desenvolvimento de sistemas de descoberta de conhecimento em banco de dados, que inclui formalismo;
- A descrição de uma forma de mapeamento de modelos UML para E-LOTOS;
- A especificação de um ambiente interativo de implementação desses sistemas, que possibilitará: a tradução automática de modelos UML para E-LOTOS, a integração de diferentes bancos de dados, a aplicação de várias técnicas de mineração de dados e a utilização de características de agentes inteligentes e móveis no desempenho de suas tarefas;
- A integração das seguintes tecnologias: modelagem orientada a objetos, métodos formais, técnicas de mineração de dados e agentes, aliando características e vantagens de cada uma no modelo integrado para o desenvolvimento de sistemas de descoberta de conhecimento.

7.2 TRABALHOS FUTUROS

As sugestões para trabalhos futuros são:

- Reavaliação do modelo de dados do metadados;
- Definição de estratégias para a implementação dos agentes móveis e inteligentes necessários ao desempenho das tarefas do ambiente, tais como: busca a bancos de dados espalhados por redes de computadores, escolha da técnica de mineração de dados mais adequada para determinado tipo de investigação e de dados e análise de resultados;
- Definição de esquemas de segurança e privacidade das tabelas do metadados e do *data set*;

- Implementação completa do ambiente ADesC, utilizando as características de agentes inteligentes e móveis;
- Desenvolvimento de componentes que implementam outras técnicas de mineração de dados;
- Aplicação da metodologia proposta em novos estudos, tomando como base o trabalho realizado com a pós-graduação brasileira;
- Desenvolvimento de outros sistemas de descoberta de conhecimento em banco de dados para atender outras áreas do conhecimento, utilizando a metodologia MeDesC e o ambiente ADesC, em parceria com outros setores da UEM (Universidade Estadual de Maringá) e com organizações da região de Maringá;
- Construção de uma ferramenta de simulação e validação de especificações de sistemas em E-LOTOS;
- Desenvolvimento de ferramenta de tradução de UML para E-LOTOS, de forma a permitir formalização de sistemas especificados segundo UML;
- Inclusão de módulos com novas tendências na área de mineração de dados (ex: algoritmos híbridos, refinamento do conhecimento extraído, etc.);
- Estudo de outras técnicas de formalização aplicadas ao processo de descoberta de conhecimento;
- Aprofundamento do modelo de repositório de dados do ADesC, de forma a compatibilizá-lo com outros modelos de implantação de *data warehouse*, por exemplo, (Kimball, 1996);
- Estudo da utilização do repositório de metadados formado por sucessivas análises KDD, visando estudos em gestão do conhecimento organizacional.

8 REFERÊNCIAS BIBLIOGRÁFICAS

AGRAWAL, R., SRIKANT, R. Fast algorithms for mining association rules. 1994.

AGRAWAL, R. et al. Automatic subspace clustering of high dimensional data for data mining applications. In: *SIGMOD Conference*, 1998: 94-105.

ARAÚJO, D.L.A.; LOPES, H.S.; FREITAS, A.A. A parallel genetic algorithm for rule Discovery in large databases. In: *IEEE Systems, Man and Cybernetics Conf.*, v. III, 940-945. Tokyo, Oct. 1999.

AZEVEDO, F.M.; BRASIL, L.M.; OLIVEIRA, R.C.L. Redes neurais com aplicações em controle e em sistemas especialistas. Visual Books, 2000.

BATES, B.W. et al. Formalizing fusion object oriented analysis models. In: *First IFIP workshop on Formal Methods for Open Object-Based Distributed Systems*, 1996.

BECHER, J.D.; BERKHIN, P. & FREEMAN, E. Automating Exploratory Data Analysis for Efficient Data Mining. In: *KDD 2000*, Boston MA USA, 424-429, 2000.

BERRY, M.J.A.; LINOFF, G. Data mining techniques. John Wiley & Sons, Inc. 1997.

BOOCH, G., RUMBAUGH, J., JACOBSON, I. The unified modeling language user guide, Addison Wesley, 1999.

BOLOGNESI, T.; BRINKSMA, E. Introduction to the ISO specification language LOTOS. In: *Computer Networks and ISDN Systems*, 14 (1987) 25-59.

BRAGA, A.P.; LUDERMIR, T.B.; CARVALHO, A.C.P.L.F. Redes neurais artificiais: teoria e aplicações, Livros Técnicos e Científicos Editora S.A., 2000.

CALCOTE, S. CyberAgents. URL:http://www.echo-strategies.com/INSIGHT/ii020596.html, 1998.

CAPES. Aplicação coleta de dados 5.0 – Documentação técnica. URL: http://www.capes.gov.br, 1999.

CAPES. Coordenação de Aperfeiçoamento de Pessoal de Nível Superior. URL:http://www.capes.gov.br, 2000

CHEN, M.-S.; HAN, J.; YU, P.S. Data mining: an overview from database perspective. TKDE 8(6): 866-883, 1996.

CHEUNG, A.Y.; CHAN, S.L. A systemic approach to define agents. In: *IRMA International Conference*, 2000, p. 207-211.

CLARKE, E.M.; WING, J.M. Formal methods: state of the art and future directions. Carnegie Mellon University. ACM Inc. URL: http://www.cs.cmu.edu/afs/cs/project/calder/www/acm.html, 1997.

COLLIER, K.; CAREY, B.; SAUTTER, D.; MARJANIEMI, C. A methodology for evaluating and selecting data mining software. In: *32nd Hawaii International Conference on System Sciences*, 1999.

CRATOCHVIL, A. Data mining techniques in supporting decision making. Master thesis, Universiteit Leiden, 1999.

CRISP-DM (CRoss Industry Standard Process for Data Mining). CRISP-DM Process Model. URL: http://www.crisp-dm.org/process2.htm, 2001.

DataMind Technology Center. Agent network technology. URL:http://datamindcorp.com/paper_agetnetwork.html, 1998.

DIAS, M.M. et al. Data warehouse – presente e futuro. In: *Revista Tecnológica*, No. 7, 1998, p. 59-73.

DOMINGOS, M.; CAMARGO, M.S. Uma introdução à nova proposta de padrão da ISO a TDF E-LOTOS com comparações a LOTOS. *Relatório Técnico do Departamento de Informática e Estatística da UFSC*, URL: http://www.inf.ufsc.br/cpgcc, 1997.

FALSARELLA, O. et al. Um serviço de mobilidade para agentes baseados em plataformas CORBA. In: *Revista do Instituto de Informática*, PUCCAMP, Campinas, V.4, n.2, p. 58-66, julho/dezembro/1996.

FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery: an overview. In: Advances in Knowledge Discovery and Data Mining, p. 1-34. AAAI Press, Menlo Park, CA, 1996.

FELDENS, M.A.; MORAES, R.L.; PAVAN, A.; CASTILHO, J.M.V. Towards a methodology for the discovery of useful knowledge combining data mining, data

warehousing and visualization. In: *XXIV CLEI (Conferência Latino-Americana de Informática)*. Quito, Equador, URL:http://jacui.inf.ufrgs.br/~feldens/clei98.html, 1998.

FREITAS, A.A. Data mining – mineração de dados. Apostila de curso ministrado no CEFET-PR, DAINF, 1998.

FREITAS, A.A. A genetic algorithm for generalized rule induction. In: *R. Roy et al. Advances in Soft Computing – Engineering Design and Manufaturing, 340-353. (Proc. WSC3, 3rd On-Line World Conference on Soft Computing, hosted on the Internet, July 1998.)* Springer-Verlag, 1999.

FREITAS, A.A. Understanding the crucial differences between classification and discovery of association rules - a position paper. In: *ACM SIGKDD Explorations*, 2(1), 65-69. ACM, 2000.

General Magic. Tabriz. URL: http://www.genmagic.com/tabriz, 1996.

GOEBEL, M.; GRUENWALD, L. A survey of data mining and knowledge discovery software tools. In: *SIGKDD Explorations*, June 1999.

GOLDBERG, D.A. Genetic algorithms in search, otimization, and machine learning. Addison-Wesley, Reading, MA, 1989.

GONÇALVES, A.L. Utilização de técnicas de mineração de dados na análise dos grupos de pesquisa no Brasil. Dissertação de Mestrado do Curso de Pós-Graduação em Engenharia de Produção da Universidade Federal de Santa Catarina, 2000.

GRAY, R.S. Agent tcl: A flexible and secure mobile-agent system. Technical Report PCS-TR98-327, Dartmouth College, Computer Science, Hanover, NH, January 1998.

GROTH, R. Data mining. Prentice Hall, Inc. 1998.

HARRISON, T.H. Intranet data warehouse. Editora Berkeley, 1998.

HAYKIN, S. Redes neurais: princípios e prática. Bookman, 2001.

HEILMANN, K. et al. Intelligent agents: a technology and business application analysis. URL: http://www.haas.berkeley.edu/~heilmann/agents/index.html, 1995.

HILLIS, D.B. Using a genetic algorithm for multi-hypothesis tracking. In: 9th International Conference on Tools with Artificial Intelligence (ICTAI'97), 1997.

HIPP, J.; GÜNTZER, U.; NAKHAEIZADEH, G. Algorithms for association rule mining – a general survey and comparison. In: *ACM SIGKDD*, July/2000, Vol. 2, No. 1, p. 58-64.

HOLSHEIMER, M.; KERSTEN, M.; MANNILA, H.; TOIVONEN, H. A perspective on databases and data mining. In: *First International Conference on Knowledge Discovery and Data Mining*, pages 447-467, AAAI Press, Menlo Park, CA. 1996.

HSU, W.; LEE, M.L.; LIU, B.; LING, T.W. Exploration Mining in Diabetic Patients Databases: Findings and Conclusions. In: *KDD 2000*, Boston MA USA, 430-436, 2000.

Hycones Information Technology. AIRA – Data mining tool. URL:http://www.hycones.com.br/portuguese.aira.htm, 1998.

IBM. Intelligent Miner.URL:http://scanner-group.mit.edu/DATAMINING/Datamining/ibm.html,1997.

INMON, W.H. Como construir o Data Warehouse. Editora Campus. 1997.

ISL Decision Systems Inc. Clementine – visual tools. URL:http://www.isl.co.uk/toolkit.html, 1996.

ISOFT SA. Alice 5.1. URL: http://www.alice-soft.com, 1998.

ISO/IEC JTC1/SC21/WG7. Final commite draft on Enhancements to LOTOS. Editor: "Enhancement to LOTOS" (1.21.20.2.3). 1997.

ISO-OSI. Information Systems Processing – Open Systems Interconnection – LOTOS. Tchnical Report, 1987.

KIMBALL, R. The data warehouse toolkit – practical techniques for building dimensional data warehouses. John Wiley & Sons, Inc. 1996.

KLEMETTINEN, M.; MANNILA, H.; TOIVONEN, H. A data mining methodology and its application to semi-automatic knowledge acquisition. In: *DEXA Workshop* 1997, p. 670-677.

KNAPIK, M.; JOHNSON, J. Developing intelligent agents for distributed systems. McGraw-Hill, 1998.

KOSALA, R.; BLOCKEEL, H. Web mining research: a survey. In: *ACM SIGKDD*, July/2000, Vol. 2, No. 1, p. 1-15.

LANGE, D.B. & CHANG, D.T. IBM aglets workbench: programming mobile agents in java. URL: http://aglets.trl.ibm.co.jp/whitepaper.html, 1996.

LANGE, D.B. & CHANG, D.T. IBM Aglets Workbench. URL:http://aglets.trl.ibm.co.jp/whitepaper.html, 1998.

LANGE, D.B. Java aglet application programming interface. URL:http://aglets.trl.ibm.co.jp/JAAPI-whitepaper.html, 1998.

LANS, R.V. O que é data mining e uma análise do mercado de produtos. In: *8º Congresso Nacional de Novas Tecnologias e Aplicações em Banco de Dados*. 1997.

LIU, A.; OFFUTT, A.J.; HO-STUART, C.; SUN, Y.; OHBA, M. SOFL: A formal engineering methodology for industrial applications. In: *IEEE Software Engineering*, Vol. 24, No. 1, January, 1998, p. 24-45.

LOH, S.; WIVES, L.K.; OLIVEIRA, J.P.M. Concept-based knowledge Discovery in texts extracted from the web. In: *ACM SIGKDD*, July/2000, Vol. 2, No. 1, p. 29-39.

MA, Y.; LIU, B.; WONG, C.K. Web for data mining: organizing and interpreting the discovered rules using the web. In: *ACM SIGKDD*, July/2000, Vol. 1, No. 1, p.16-23.

MANNILA, H. Data mining: machine learning, statistics, and databases. In: *Eight International Conference on Scientific and Database Management*, 1996, p.1-8.

MANNILA, H. Methods and problems in data mining. In*: International Conference on Database Theory*, Delphi, Greece, January 1997.

Megaputer Intelligence Ltd. PolyAnalyst 3.5. http://www.megaputer.com, 1998.

MEHTA, M. et al. SLIQ: a fast scalable classifier for data mining. EDBT, 1996: 18-32.

MILNER, R. A calculus of communicating systems. Lecture and Notes in Comp. Science, No. 92, Springer Verlag, 1980.

Mitsubishi Electric Information Technology Center America. Mobile agents: a new software paradigm for distributed application development. URL: http//www.meitca.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.html, 1997.

MOBASHER, B.; COOLEY, R.; SRIVASTAVA, Automatic personalization based on web usage mining. In: *Communications of the ACM*, Vol. 43, No. 8, 2000, p. 142-151.

MOREIRA, A.M.D.; CLARK, R.G. Using rigorous object-oriented analysis. In: *VII Simpósio Brasileiro de Engenharia de Software*, 1993, p.238-252.

NeoVista Solutions Inc. Decision Series 3.0. http://www.neovista.com, 1998.

NWANA, H.S. Software agents: an overview. URL:http://www.cs.umbc.edu/agents/introduction/ao/, 1996.

PRESSMAN, R. Engenharia de Software. Makron Books, São Paulo, 1999.

PYLE, D. Data preparation for data mining. Morgan Kaufmann Publishers. 1999.

ROMÃO, W. et al. Extração de regras de associação em C & T: o algoritmo Apriori. In: *XIX Encontro Nacional em Engenharia de Produçao, V ICIE - International Congress of Industrial Engineering*. Rio de Janeiro, 1999.

RUMBAUGH, J. et al. Modelagem e projetos baseados em objetos. Editora Campus, 1994.

RUSHBY, J. Formal methods and the certification of critical systems. Computer Science Laboratory SRI International, Menlo Park CA 94025 USA. Technical Report CSL-93-7, December 1993.

SARAWAGI, A.; NAGARALU, S.H. Data mining models as services on the internet. In: *ACM SIGKDD*, July/2000, Vol 2, No. 1, p. 24-28.

SHAFER, J.C. et al. SPRINT: a scalable parallel classifier for data mining . VLDB, 1996: 544-555.

Silicon Graphics Computer Systems. MineSet. http://www.sgi.com/chembio/apps/datamine.html, 2000.

SILVA, E.L.; MENEZES, E.M. Metodologia da pesquisa e elaboração de dissertação. Laboratório de Ensino à Distância da UFSC, 2000.

SINNOT, R.O.; TURNER, K.J. Applying formal methods to standard development: the open distributed processing experience. Department of Computing Science and Mathematics, University of Stirling, Stirling FK9 4LA, Scotland, June, 1994.

SPILIOPOULOU, M. Web usage mining for web site evaluation. In: Communications of the ACM, Vol. 43, No. 8, 2000, p.127-134.

Two Crows Corporation. Introduction to data mining and knowledge discovery. Third Edition, 1999.

VIVEROS, M.S.; NEARHOS, J.P.; ROTHMAN, M.J. Applying data mining techniques to a health insurance information system. In: *22nd VLDB Conference*, Mumbai (Bombay), India, 1996.

ZHANG, T. et al. BIRCH: an efficient data clustering method for very large database. SIGMOD, 1996, p. 103-114.

WHITLEY, D. A Genetic algorithm tutorial. Technical Report, 1993.

WIRYANA, M. Information system development: an interdiscipline approach. URL: http://nakula.rvs.uni-bielefeld.de/made/aartikel/paper_issm/, 1998.

WOOLDRIDGE, M.; JENNINGS, N. Intelligent agents: theory and practice. In: *Knowledge Engineering Review*, 1995.

9 ANEXOS

9.1 A LINGUAGEM E-LOTOS

9.1.1 Introdução

LOTOS (*Language of Temporal Ordering Specification*) é uma Técnica de Descrição Formal (TDFs) geralmente aplicável a sistemas distribuídos, concorrentes e de processamento de informações (ISO-OSI, 1987) e, também, é uma das TDFs mais utilizadas nas metodologias que integram metodologias orientadas a objetos e TDFs.

A linguagem E-LOTOS é uma linguagem formal baseada em LOTOS padronizada pela ISO (*International Standard Organization*).

Este apêndice faz algumas considerações gerais sobre LOTOS e descreve as principais características da linguagem E-LOTOS.

9.1.2 LOTOS

A Técnica de Descrição Formal LOTOS foi desenvolvida pela ISO para especificação formal de sistemas distribuídos abertos e, em particular, para aqueles definidos para arquitetura de rede de computador OSI (*Open Systems Interconnection*).

A idéia básica desenvolvida em LOTOS foi que sistemas podem ser especificados pela definição de uma relação temporal entre as interações que constituem o comportamento de um sistema externamente observável (Bolognesi, 1987). Ao contrário do que o nome parece sugerir, esta técnica de descrição não é relacionada à lógica temporal, mas é baseada em métodos algébricos. Tais métodos foram introduzidos primeiro por Milner (1980), logo seguido por outros autores que relacionam esses métodos a outras teorias que são freqüentemente referenciadas como "álgebra de processos".

A TDF LOTOS inclui, também, um segundo componente que lida com a descrição de estruturas de dados e expressões de valor. A maioria dos conceitos, desses componentes, foi inspirada pela técnica de tipo de dados abstratos ACT ONE (Álgebra para Dados Abstratos).

Os critérios gerais que têm determinado a definição presente de LOTOS são (Bolognesi, 1987):

- Poder de expressão;
- Definição formal;
- Abstração;
- Estrutura.

Em LOTOS um sistema distribuído e concorrente é visto como um processo, possivelmente consistindo de vários sub-processos (Bolognesi, 1987). Um sub-processo é um processo nele próprio, de tal forma que em geral uma especificação LOTOS descreve um sistema via uma hierarquia de definições de processo.

Um processo é uma entidade hábil para executar ações internas não observáveis, e para interagir com outros processos, que formam seu ambiente. Interações complexas entre processos são construídas de unidades elementares de sincronização que são chamadas eventos ou interações (atômicas), ou simplesmente ações.

Os eventos implicam em sincronização de processos, porque os processos que interagem através de um evento participam em sua execução no mesmo instante de tempo. Tal sincronização pode envolver a troca de dados. Os eventos são atômicos porque eles ocorrem instantaneamente, sem consumir tempo. Um evento ocorre num ponto de interação, ou porta, e no caso de sincronização sem troca de dados, o nome do evento e o nome da porta coincidem.

Uma versão simplificada da linguagem, conhecida como LOTOS básico, emprega um alfabeto finito de ações observáveis. Isto porque as ações observáveis aqui são identificadas somente pelo nome da porta onde elas são oferecidas, processos em LOTOS podem ter somente um número finito de portas.

Um componente essencial de uma definição de processo é sua expressão de comportamento. Uma expressão de comportamento é construída pela aplicação de um operador (por ex., '[]') para outras expressões de comportamento. Uma expressão de comportamento pode, também, incluir instanciações de outros processos, cujas definições são fornecidas na cláusula *where*, seguindo a expressão.

9.1.3 E-LOTOS

A linguagem E-LOTOS (ISO-IEC, 1997) é baseada em LOTOS e possui alguns aprimoramentos relacionados basicamente com a declaração de tipos de dados e com a

associação do tempo de forma explícita na especificação do comportamento de sistemas. Além disso, E-LOTOS apresenta uma linguagem modular que foi criada para suprir as deficiências e limitações de LOTOS na definição de módulos.

Ainda de acordo com (ISO-IEC, 1997), "E-LOTOS é, como LOTOS, uma linguagem que permite uma rica variedade de estilos de especificações que podem ser usadas para modelar diferentes aspectos do processo de projeto".

E-LOTOS é uma linguagem em dois níveis. O nível superior trata-se da linguagem modular (*Modular* Language). O nível inferior é representado pela linguagem base (*Base Language*).

Uma especificação descrita na linguagem base é uma seqüência de declarações, podendo ser estruturada em nível modular. Os tipos de declarações são:

- Declarações de Tipos de Dados;
- Declarações de Funções; e
- Declarações de Processos.

Declaração de Tipos de Dados

Uma declaração de tipo pode ser uma declaração de um sinônimo de tipo ou uma declaração de tipo de dado. Um sinônimo de tipo declara um novo identificador de tipo para um tipo existente.

A linguagem E-LOTOS permite definir os tipos de dados de uma forma mais amigável do que LOTOS; sendo possível, também, a definição de tipos de dados concretos, descrevendo como os valores de dados são representados e como alguns procedimentos associados operam sobre eles. Os tipos de dados abstratos são representados em nível de linguagem modular.

Em E-LOTOS tem-se um tipo pré-definido que é crucial na definição da semântica temporal; o tipo *Time* (Domingos, 1997). Este tipo pode ser considerado, de forma excludente, sobre dois conjuntos de domínio: os inteiros (tempo discreto) ou os racionais (tempo denso).

As ações que representam a passagem do tempo são denotadas na semântica da linguagem pela notação $\varepsilon(d)$, onde d representa o elemento do domínio de tempo *Time* significando a passagem de um tempo d no sistema de transição.

Declaração de Funções

Uma função é apenas um processo que executa somente uma ação de terminação (δ), ou seja, não existe nenhuma outra ação possível. Assim, quando esta ação ocorrer, os dados resultantes serão passados para o processo no qual a função está envolvida.

Portanto, uma declaração de função define uma nova função, sendo que uma função poderá ser utilizada somente nas expressões de dados, não podendo representar o comportamento de um sistema.

Declaração de Processos

Declarações de processos são muito similares a declarações de funções: eles têm listas de parâmetros de entrada e de saída e uma lista de parâmetros de exceção tipados.

Entretanto, existem diferenças importantes entre funções e processos: processos podem ter comportamento tempo-real e podem se comunicar através de portas.

9.1.3.1 Expressões de Comportamento

A linguagem E-LOTOS possui um conjunto de expressões de comportamento e um conjunto de operadores, associados às expressões, que permitem representar o comportamento do sistema. As expressões de comportamento de E-LOTOS e os operadores correspondentes são apresentados na tabela I.1, onde *B*, *B1* e *B2* representam expressões de comportamento.

9.1.3.2 A Linguagem Modular

A idéia por trás da linguagem modular é fazer com que os módulos utilizados na parte de dados sejam os mesmos utilizados na parte comportamental, ou seja, será permitido definir, dentro de um módulo, tipos de dados, funções e processos.

Utilizando a linguagem modular, um sistema pode ser especificado através de uma seqüência dos seguintes tipos de declarações:

- Declaração de Módulos;
- Declaração de Interfaces; e
- Declaração de Módulos Genéricos.

Tabela 9.1: Sintaxe Resumida de Expressões de Comportamento em E-LOTOS

Nome	**Sintaxe resumida**
Inação e bloqueio no tempo	stop e block
Ações	
- ação interna (*i*)	i; *B*
- ação de *delay* (ε)	wait(<*expressão de dados*>); *B*
- ação de terminação (δ)	exit[(<*expressão de valor*>)] e exit(any<*expressão de tipo*>)
- ação representando uma exceção	signal <*identificador de exceção*> [<*exp. dados*>)] e raise <*identificador de exceção*> [<*exp. dados*>)]
Composição seqüencial	*B1*; *B2*
Atribuições	*P* := *E*
Desabilitação	*B1* [> *B2*
Composição paralela	
- concorrência	*B1*|[*g1,...,gn*]|*B2*
- sincronização completa	*B1*||*B2*
- entrelaçamento	*B1*|||*B2*
- paralelismo genérico	par ... endpar
Escolha	*B1* [] *B2* e choice ... endch
Tratamento de exceção	trap ... endtrap
Expressões case e if-then-else	case ... endcase e if *E* then *B1* else *B2* endif
Declaração de variáveis locais	local var <*variáveis locais*> [init *B1*] in *B2* endloc
Ocultação	hide [<*portas*>] in *B* endhide
Renomeação	rename ... in *B* endren
Instanciação de processo	p[<*portas*>] [(<*expressão de dados*>)] [<*exceções*>]
Iterações	loop ... endloop

Declaração de Módulos

Um módulo é uma seqüência de declarações de tipos, construtores, valores constantes, funções e processos, utilizando-se de todos os recursos da linguagem base. Assim, um módulo pode ser declarado da seguinte forma:

module <*identificador de módulo*> **is** <*declarações*> **endmod**

O exemplo a seguir mostra a definição do tipo *nat* em E-LOTOS (Domingos, 1997):

```
module NatNumbers is
        value 0:nat endval
        type nat is succ(nat)
endmod
```

Neste caso, o tipo *nat* é definido como sendo o conjunto de valores: 0, *succ*(0), *succ*(*succ*(0)), ... que pode denotar o conjunto dos números naturais (0, 1, 2, ...).

Para adicionar operadores (por ex.: o operador "+"), pode-se definir um novo módulo contendo funções que descrevem os operadores. Este novo módulo utilizará a definição do tipo *nat* do módulo *NatNumbers* através da cláusula "*import*".

```
module ExtNatNumbers import NatNumbers is
   function + (x : nat, y : nat) : nat is
        local z : nat in
                case x is
                  0 → y
                | succ(z) → succ(z + y)
                endcase
        endloc
   endfun
endmod
```

É possível definir um módulo que contenha uma declaração externa, permitindo assim a definição deste módulo através de uma outra linguagem de especificação (ou de implementação). O exemplo abaixo define a implementação externa para o módulo *nat*.

```
module ExtNat is external endmod
```

Declaração de Interface

Intuitivamente, uma interface é um tipo de módulo. Enquanto uma expressão de interface especifica uma classe de módulos, uma expressão de módulo declara um módulo. A interface abaixo pode se encaixar com qualquer módulo que contenha, pelo menos, o tipo identificado como *nat* e as definições encontradas nesta interface. Assim

o módulo do exemplo anterior poderia ser redefinido como o módulo *ExtNatNumbers* apresentado anteriormente.

```
interface Nat is
        type nat is
                0
          | succ(nat)
        endtype
endint
```

Declaração de Módulos Genéricos

A declaração de módulos genéricos possibilita a construção de especificações que permitam a reutilização de código. Assim, diversos componentes de uma biblioteca padrão podem ser definidos por um módulo genérico.

A utilização de um módulo genérico pode ser feita dentro de um módulo normal, onde é possível alterar os elementos da interface associada ao módulo genérico, ou seja, dado um módulo genérico:

```
generic GenericList (Eq : EqType) : List is
    ...
endgen
```

então, um módulo pode utilizar esta generalização alterando os seus parâmetros, como a seguir:

```
module ListNat : [List renaming (E for nat)] is
        GenericList (Natural renaming (nat for E))
    ...
endmod
```

sendo feita a alteração dos parâmetros através da primitiva **renaming**.

9.1.3.3 Especificação das Restrições de Tempo

Em E-LOTOS são previstas várias formas de se especificar as restrições de tempo, como a seguir (Domingos, 1997):

1. Restrições de tempo: podem estar relacionadas à realização de uma ação dentro de um intervalo de tempo [*Min*, *Max*], que pode ser representada por:

```
process P : exit(none) is
        g@t[t<Max] start(Min); Q
endproc
```

2. Retardo (*delay*): um retardo de $t \in Time$ unidades de tempo imposto a um processo P pode ser representado por:

```
process P : exit(none) is
        wait(t); Q
endproc
```

3. Processo periódico: seja Q um processo e $t \in Time$ uma quantidade de tempo, então um processo P que representa o lançamento do processo Q a cada período de tempo t pode ser representado por:

```
process P:exit(none) is
    loop forever
        wait(t);Q
    endloop
endproc
```

4. *Jitter*: um *jitter* é a variação dentro de um espaço temporal $d \in Time$ que um processo periódico de período $t \in Time$ pode sofrer.

```
process R:exit(none) is
    loop forever
        is g@v[v<(t+d)]start(t);Q
    endloop
endproc
```

5. *Timeout*: sejam P e Q dois comportamentos, e $t \in Time$; então, um *timeout* é um mecanismo dependente do tempo que se comporta como P, se uma ação inicial de P ocorre até o instante t, ou como Q após o tempo t.

```
process R : exit(none) is
        P []
        wait(t);Q
endproc
```

6. *Watchdog*: este mecanismo pode ser modelado utilizando o operador de preempção, da maneira que segue:

```
process R : exit(none) is
        P [> wait(t);Q
endproc
```

9.2 DIAGRAMAS DE ESTADOS DO AMBIENTE ADesC

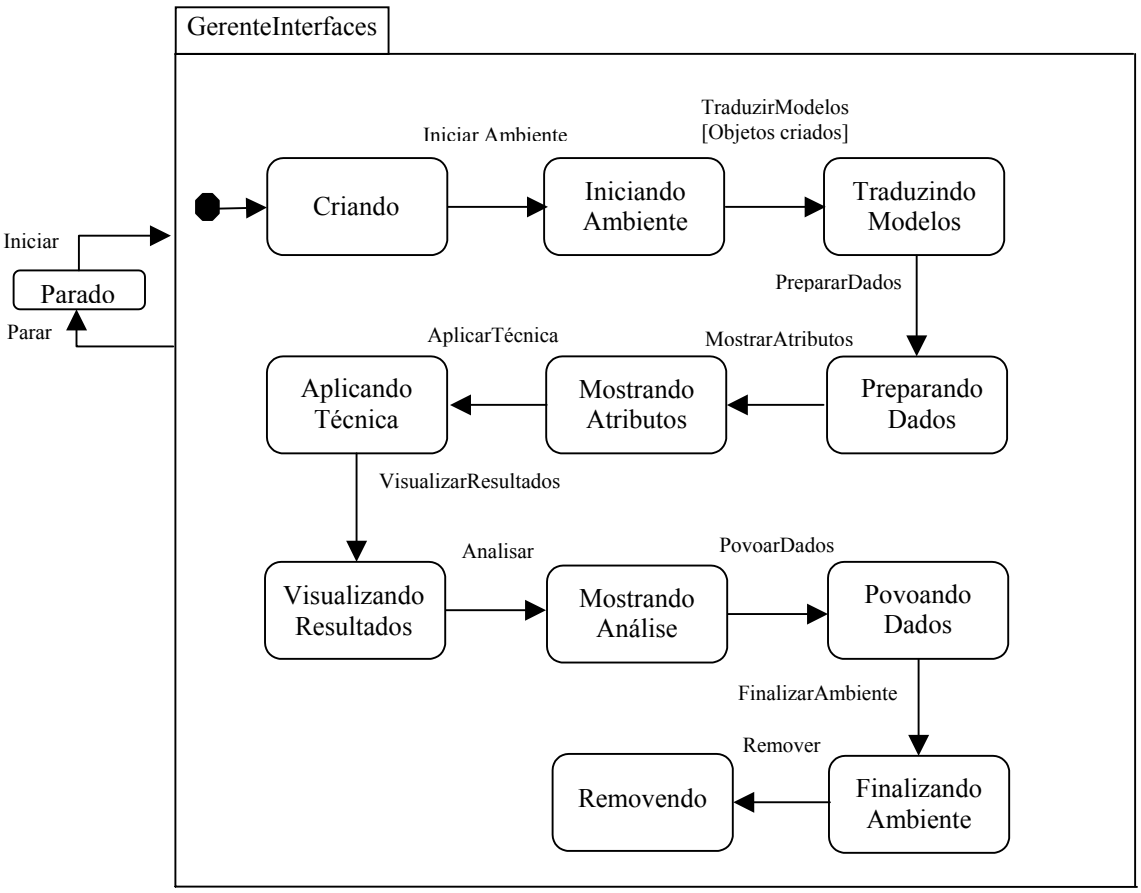


Figura 9.1: Diagrama de Estados da Classe GerenteInterfaces

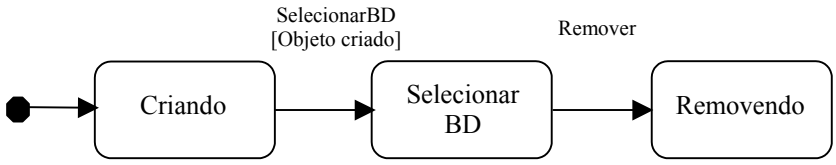


Figura 9.2: Diagrama de Estados da Classe InterfaceEntrada

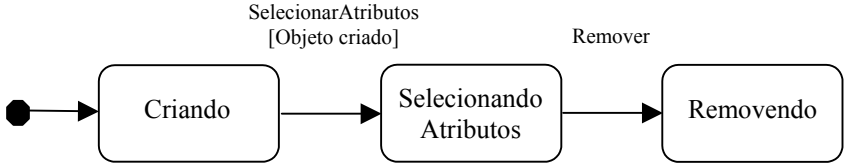


Figura 9.3: Diagrama de Estados da Classe InterfaceDados

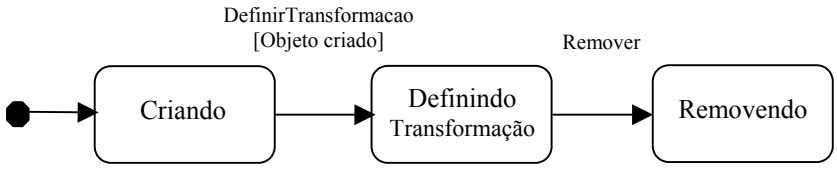


Figura 9.4: Diagrama de Estados da Classe InterfaceTransformação

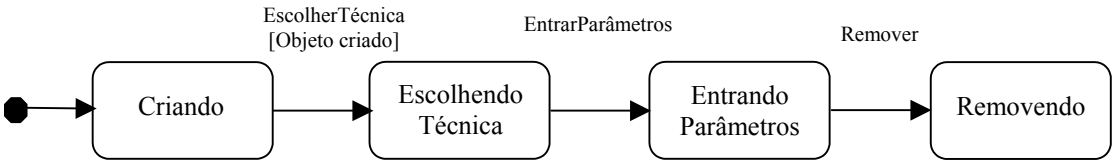


Figura 9.5: Diagrama de Estados da Classe InterfaceTécnica

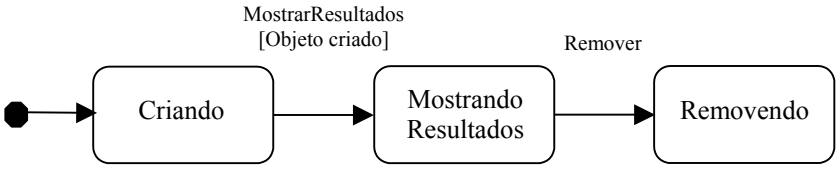


Figura 9.6: Diagrama de Estados da Classe InterfaceResultados

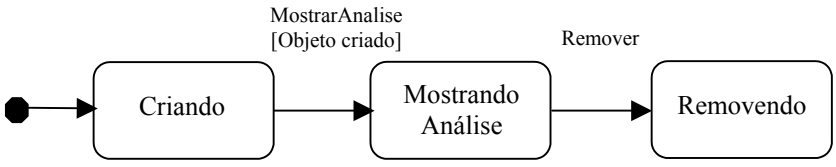


Figura 9.7: Diagrama de Estados da Classe InterfaceAnalise

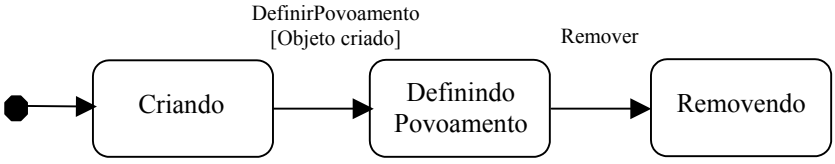


Figura 9.8: Diagrama de Estados da Classe InterfacePovoamento

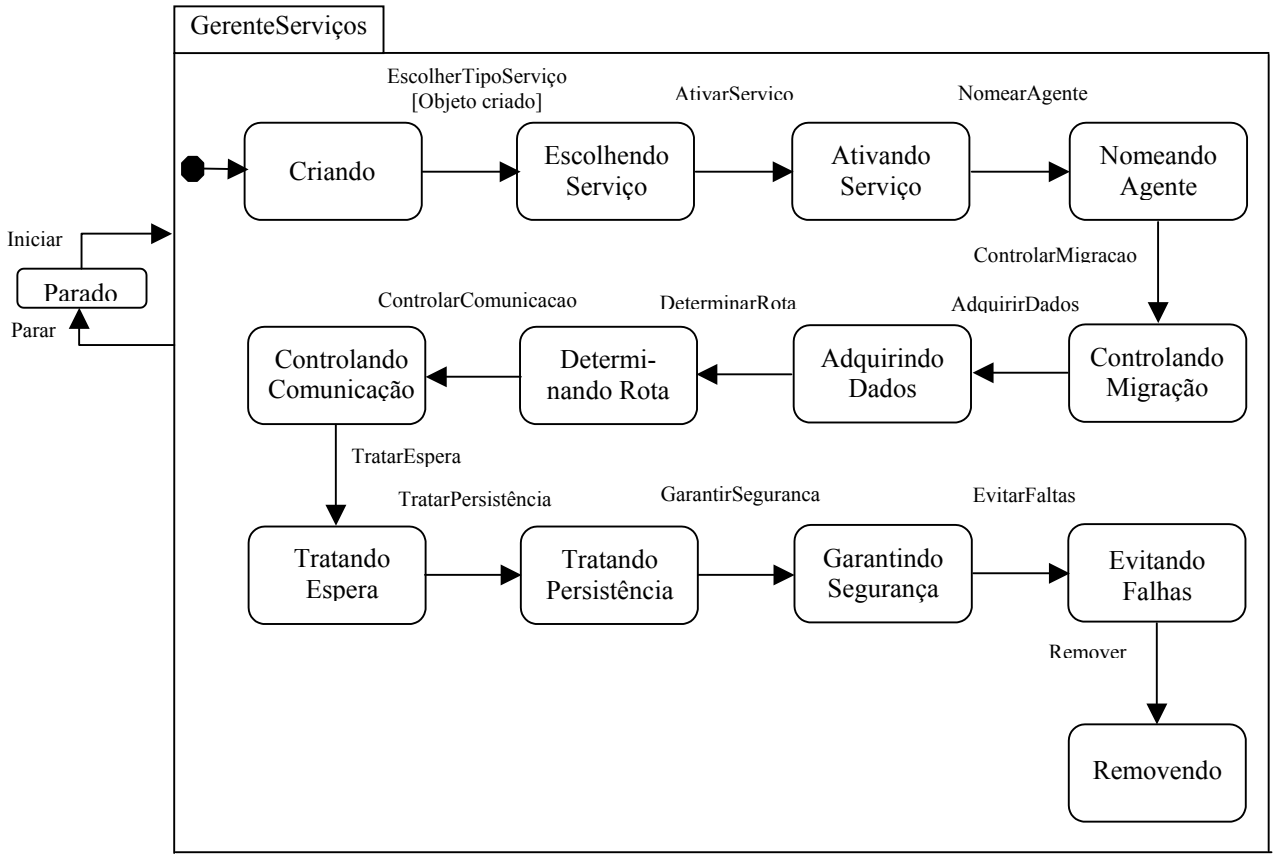


Figura 9.9: Diagrama de Estados da Classe GerenteServiços

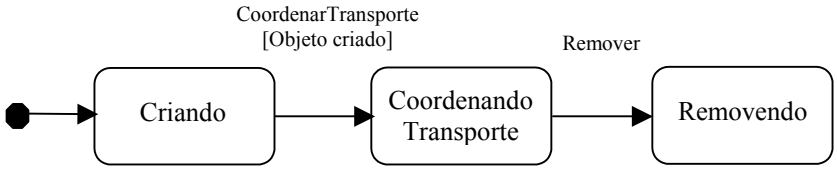


Figura 9.10: Diagrama de Estados da Classe CoordenadorTransporte

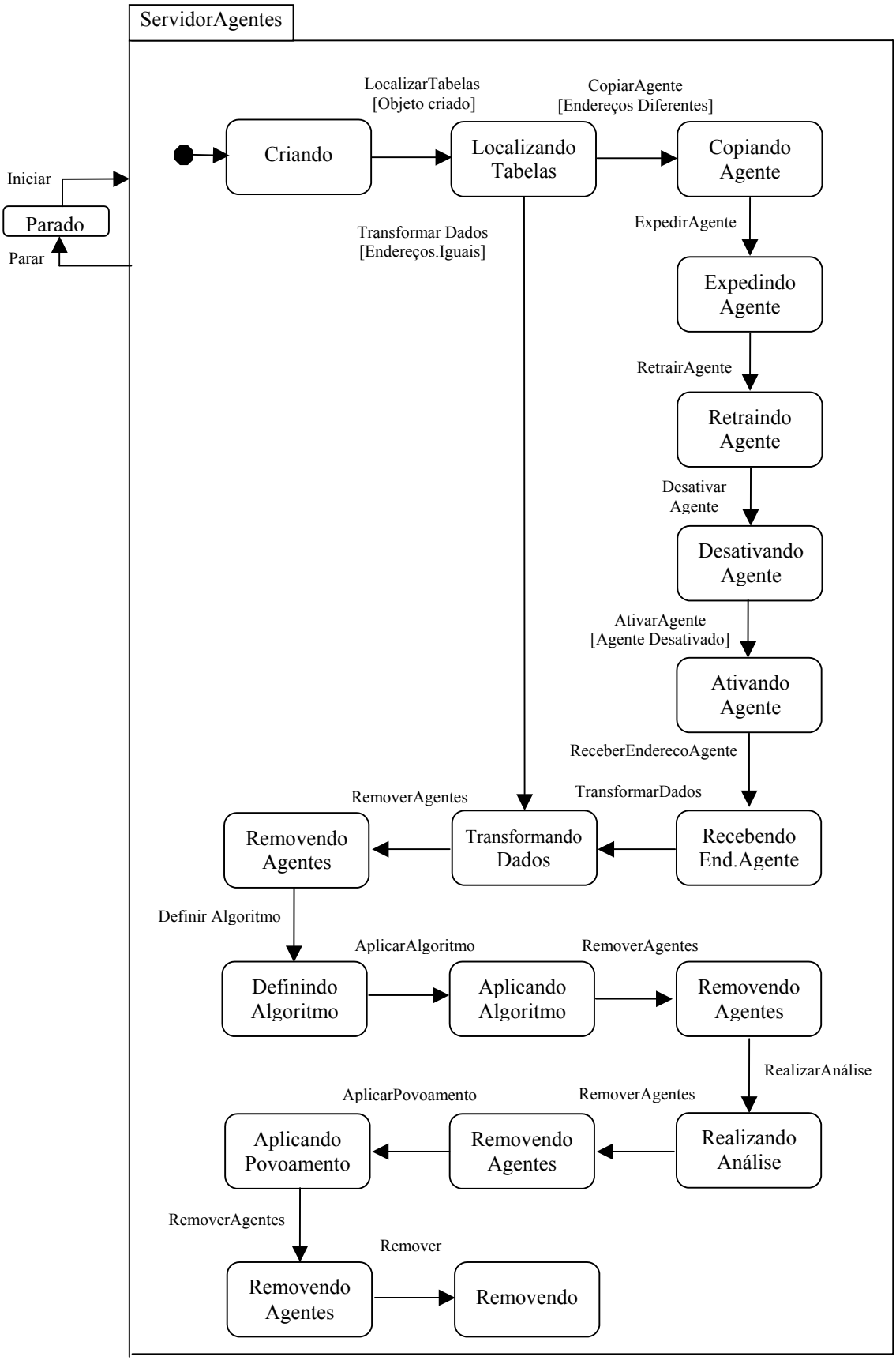


Figura 9.11: Diagrama de Estados da Classe ServidorAgente

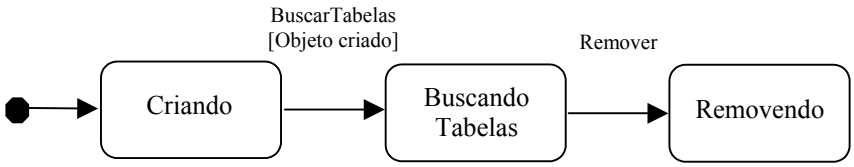


Figura 9.12: Diagrama de Estados da Classe AgenteBusca

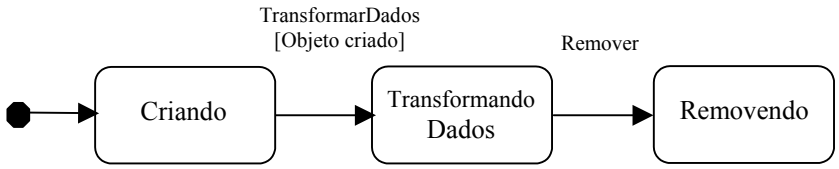


Figura 9.13: Diagrama de Estados da Classe AgenteTransformacao

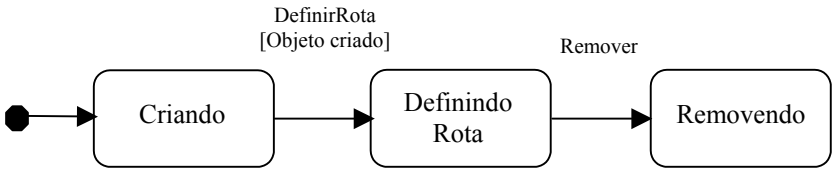


Figura 9.14: Diagrama de Estados da Classe AgenteRoteador

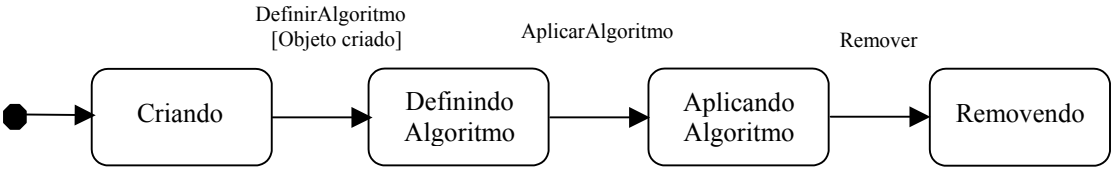


Figura 9.15: Diagrama de Estados da Classe AgenteTecnica

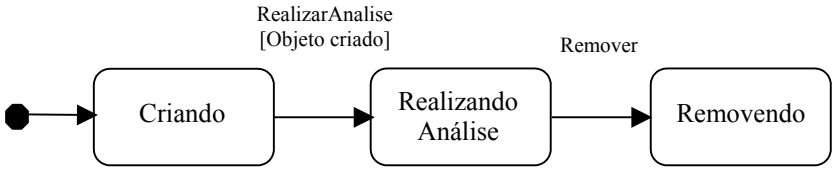


Figura 9.16: Diagrama de Estados da Classe AgenteAnalise

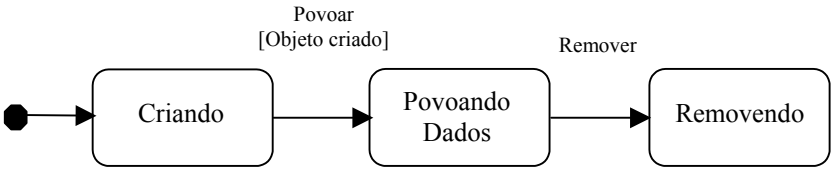


Figura 9.17: Diagrama de Estados da Classe AgentePovoamento

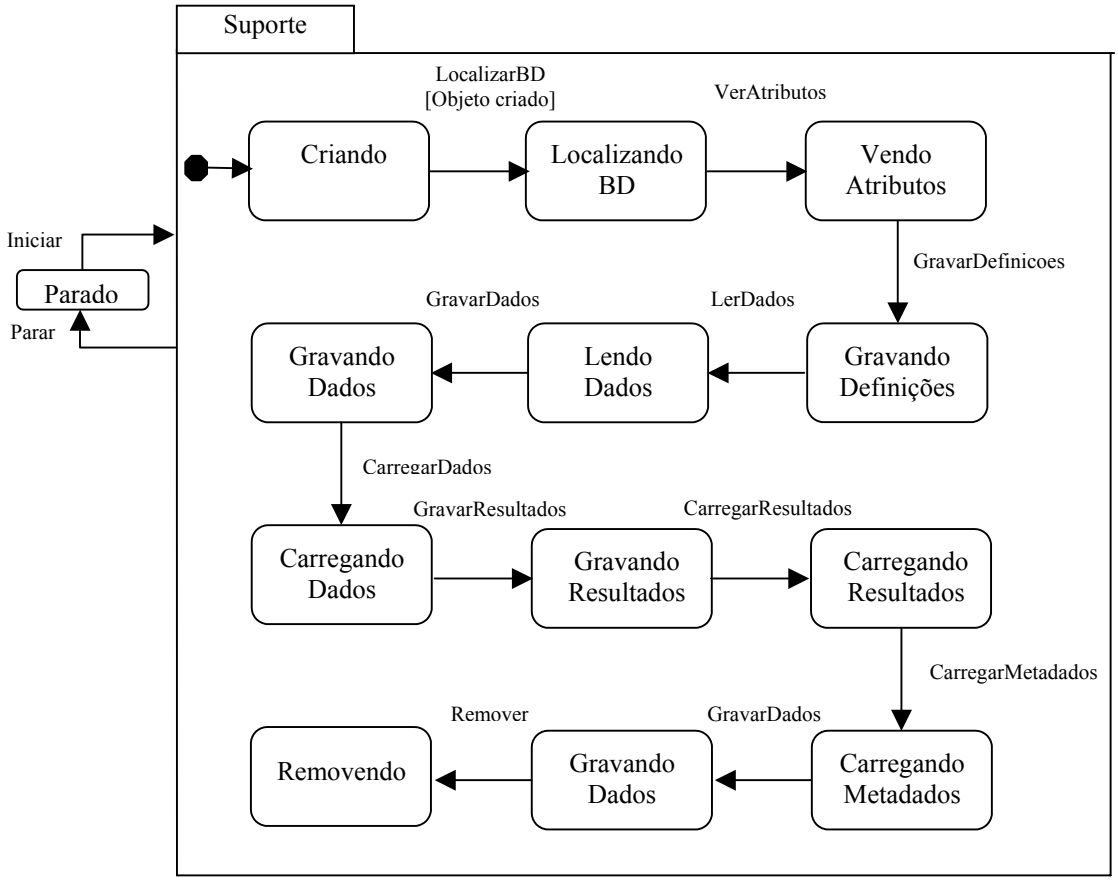


Figura 9.18: Diagrama de Estados da Classe Suporte

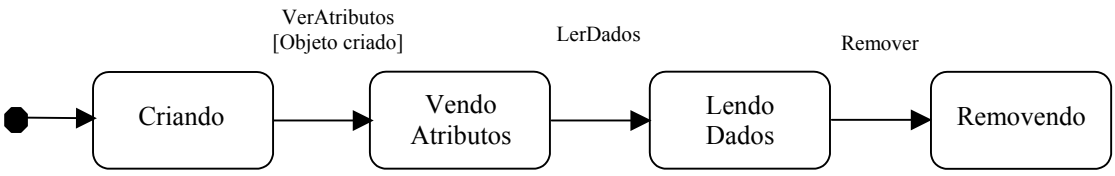


Figura 9.19: Diagrama de Estados da Classe BancoDados

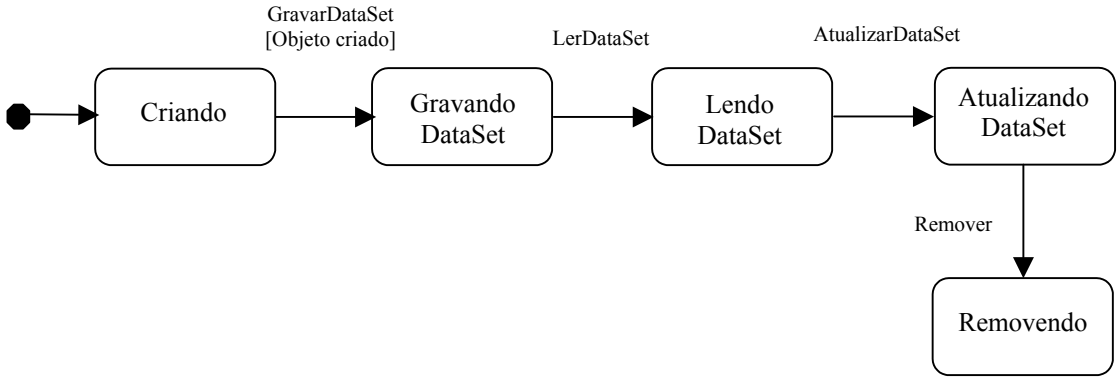


Figura 9.20: Diagrama de Estados da Classe DataSet

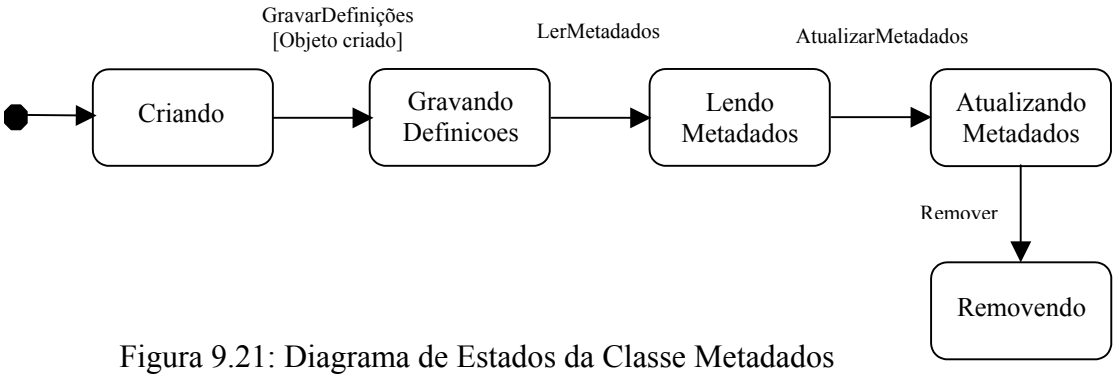


Figura 9.21: Diagrama de Estados da Classe Metadados

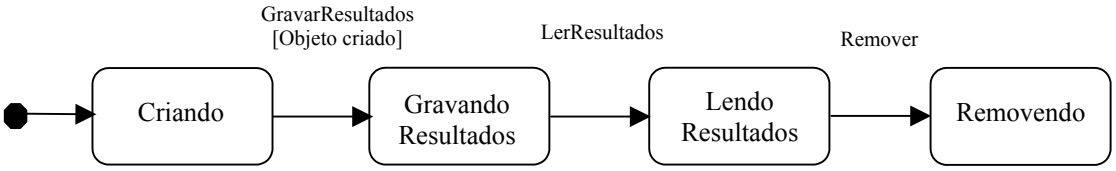


Figura 9.22: Diagrama de Estados da Classe Resultados

9.3 UM MODELO DE METADADOS

9.3.1 Introdução

Este anexo apresenta um modelo de metadados proposto para o ambiente ADesC.

A Tabela 9.2 relaciona as tabelas de banco de dados que compõem o modelo proposto e suas respectivas definições. O Diagrama Entidade-Relacionamento deste modelo é apresentado na Figura 9.23.

O modelo de metadados proposto cobre as necessidades básicas de uma ferramenta para preparação de dados, a saber:

- definição de tabelas;
- definição das regras de transformação e povoamento;
- definição da freqüência com que cada regra deve ser executada;
- documentação dos metadados.

Estas necessidades básicas são descritas nos itens a seguir.

9.3.2 Definição das Tabelas

As tabelas Base, Tabela e Coluna armazenam a estrutura das tabelas operacionais e históricas envolvidas na preparação dos dados. Uma Base de dados contém Tabelas, as quais são formadas por Colunas. Cada coluna armazena dados de tipo e tamanho (nem sempre) pré-determinados. O resultado do relacionamento entre as tabelas Base, Tabela e Coluna é um mapeamento direto dos elementos envolvidos na preparação de dados.

Uma Tabela é classificada por Indice, os quais são formados por elementos IndColuna, ou seja, por uma seqüência de colunas pertencentes à tabela em questão. Cada coluna pode ser classificada em ordem ascendente ou descendente. A chave de indexação pode ou não ser única.

9.3.3 Definição das Regras de Transformação e Povoamento

Como a definição das regras de transformação e povoamento não é uma tarefa trivial, julgou-se necessário criar um vocabulário contendo expressões (e seus argumentos) permitidas nas regras de transformação e povoamento.

As regras de transformação e povoamento são a essência deste modelo. Uma Tabela possui pelo menos uma regra de Povoamento, que contém um item PovColuna para cada coluna que possua um relacionamento com o item de Tabela em questão. PovColuna é uma generalização das transformações e/ou povoamentos que deve ser aplicada à coluna em questão. O Modelo prevê três tipos de transformações:

- **Atribuição direta**: o dado é copiado da base operacional sem nenhuma alteração;
- **Discretização** ou mapeamento: uma informação contínua é dividida em categorias, ou um item discreto pode ter sua representação mapeada para uma outra forma, coerente com a representação adotada pela base de dados histórica;
- **Expressões**: são capazes, teoricamente, de implementar qualquer tipo de transformação não coberta pelos itens anteriores e por outros que poderiam ser adicionados ao modelo. As expressões serão vistas em maior detalhe a seguir.

9.3.4 Expressões

As regras de transformação e povoamento não podem ser definidas somente em termos de atribuição direta e discretização. A implementação de uma base de dados histórica depende de informações que, para serem obtidas, exigem que algumas operações mais complexas sejam realizadas sobre os dados operacionais, tais como:

- tratamento e manipulação de string;
- cálculo de médias;
- consultas SQL;
- expressões condicionais;

A solução para operações deste tipo exige uma estrutura semelhante a uma linguagem de programação: as regras de transformação e povoamento são representadas por sentenças, que possuem como argumentos os meta-elementos de dados operacionais

e históricos. Estas sentenças (bem como o vocabulário da linguagem) são armazenadas junto aos metadados.

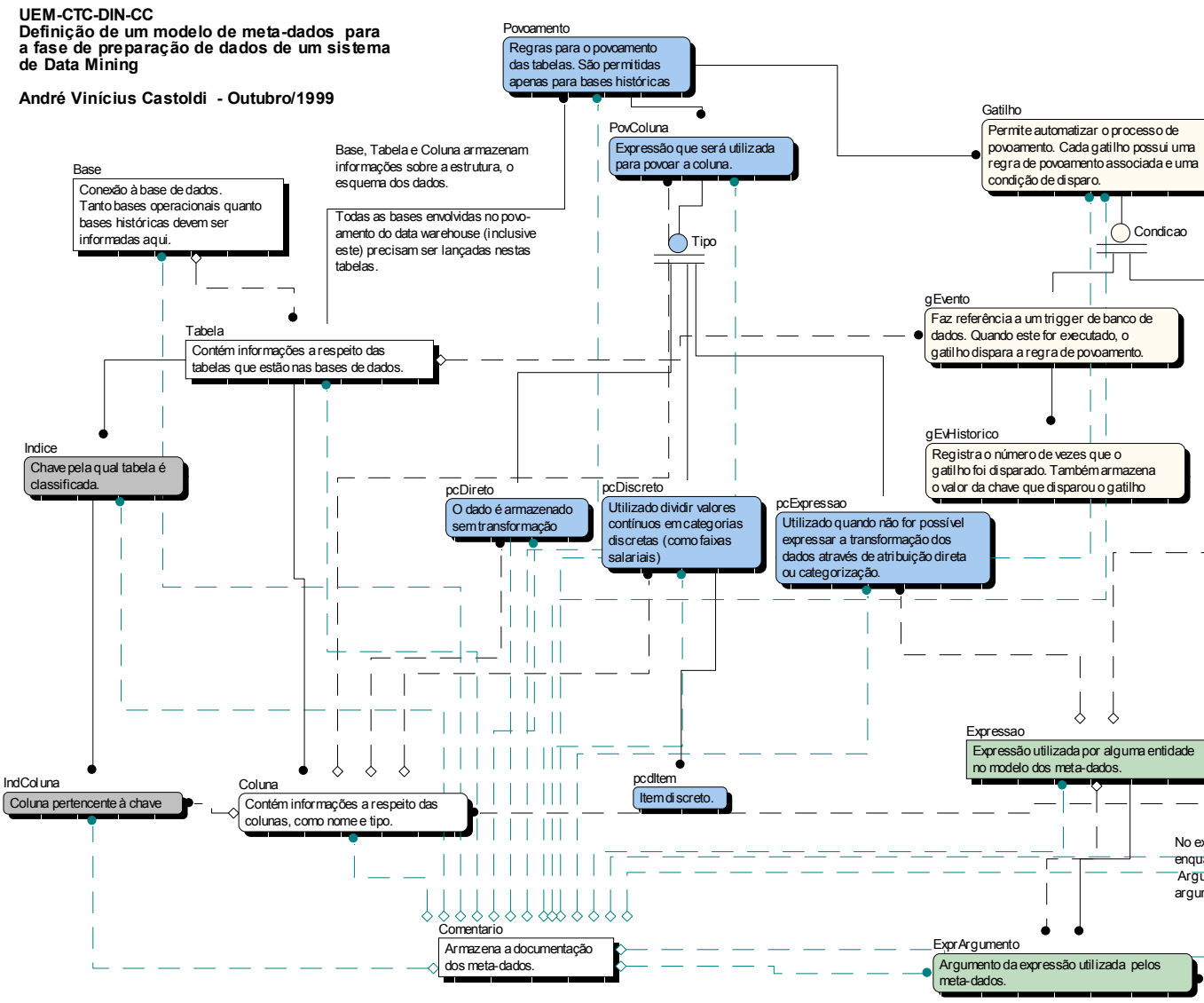
A execução destas sentenças exige um compilador. O modelo baseou-se na linguagem LISP, onde todos os operadores são pré-fixados e os argumentos são bem delimitados, o que reduz o trabalho de construção dos analisadores léxicos, sintáticos e semânticos.

9.3.5 Gatilhos

O objetivo dos gatilhos é automatizar o processo de transformação e povoamento, pois, a maioria das regras é executada periodicamente. O modelo prevê gatilhos disparados por Eventos, de forma semelhante aos gatilhos implementados em SBGDs, e por Expressões Booleanas. Neste caso, o gatilho deve ser disparado quando o resultado da expressão é verdadeiro. O modelo também mantém um histórico de disparo dos gatilhos, contendo a data, hora e expressão (ou chave) que disparou o gatilho.

Tabela 9.2: Descrição das Tabelas que Compõem o Modelo de Metadados

Tabela	**Definição**
Base	Conexão à base de dados. Tanto bases operacionais quanto bases históricas devem ser informadas aqui
Tabela	Contém informações a respeito das tabelas que estão nas bases de dados.
Coluna	Contém informações a respeito das colunas, como nome e tipo.
Indice	Chave pela qual tabela é classificada.
IndColuna	Coluna pertencente à chave.
Povoamento	Regras para o povoamento das tabelas. Devem ser permitidas apenas para bases históricas.
PovColuna	Expressão utilizada para povoar a coluna.
pcDireto	O dado é armazenado sem transformação.
pcDiscreto	Utilizado dividir valores contínuos em categorias discretas (como faixas salariais).
pcdItem	Item discreto. Utilizado em conjunto com pcDiscreto.
pcExpressao	Utilizado quando não for possível expressar a transformação dos dados através de atribuição direta ou categorização.
Expressao	Expressão utilizada por alguma entidade no modelo dos metadados.
ExprArgumento	Argumento da expressão utilizada pelos metadados.
Operador	Relação dos operadores pertencentes à gramática utilizada nas expressões.
Argumento	Relação dos argumentos necessários a um operador. Todos argumentos devem ser preenchidos para que a expressão tenha um funcionamento correto.
TipoDado	Armazena os tipos de dados retornados pelas expressões e utilizados nas definições das colunas.
Comentario	Armazena a documentação dos metadados.
Gatilho	Permite automatizar o processo de povoamento. Cada gatilho possui uma regra de povoamento associada e uma condição de disparo.
gEvento	Faz referência a um trigger de banco de dados. Quando este for executado, o gatilho dispara a regra de povoamento.
gEvHistorico	Registra o número de vezes que o gatilho foi disparado. Também armazena o valor da chave que disparou o gatilho.
gExpressao	Expressão contendo a condição de disparo. O gatilho é disparado quando a Expressão atinge o valor alvo.
gExHistorico	Registra o número de vezes que o gatilho foi disparado. Também armazena o valor da expressão que disparou o gatilho.


UEM-CTC-DIN-CC
Definição de um modelo de meta-dados para a fase de preparação de dados de um sistema de Data Mining
André Vinícius Castoldi - Outubro/1999
Povoamento
Regras para o povoamento das tabelas. São permitidas apenas para bases históricas
Gatilho
Permite automatizar o processo de povoamento. Cada gatilho possui uma regra de povoamento associada e uma condição de disparo.
PovColuna
Expressão que será utilizada para povoar a coluna.
Base, Tabela e Coluna armazenam informações sobre a estrutura, o esquema dos dados.
Base
Conexão à base de dados. Tanto bases operacionais quanto bases históricas devem ser informadas aqui.
Todas as bases envolvidas no povoamento do data warehouse (inclusive este) precisam ser lançadas nestas tabelas.
Tipo
Condicao
gEvento
Faz referência a um trigger de banco de dados. Quando este for executado, o gatilho dispara a regra de povoamento.
Tabela
Contém informações a respeito das tabelas que estão nas bases de dados.
gEvHistorico
Registra o número de vezes que o gatilho foi disparado. Também armazena o valor da chave que disparou o gatilho
Indice
Chave pela qual tabela é classificada.
pcDireto
O dado é armazenado sem transformação
pcDiscreto
Utilizado dividir valores contínuos em categorias discretas (como faixas salariais)
pcExpressao
Utilizado quando não for possível expressar a transformação dos dados através de atribuição direta ou categorização.
Expressao
Expressão utilizada por alguma entidade no modelo dos meta-dados.
IndColuna
Coluna pertencente à chave
Coluna
Contém informações a respeito das colunas, como nome e tipo.
pcdItem
Item discreto.
Comentario
Armazena a documentação dos meta-dados.
ExprArgumento
Argumento da expressão utilizada pelos meta-dados.

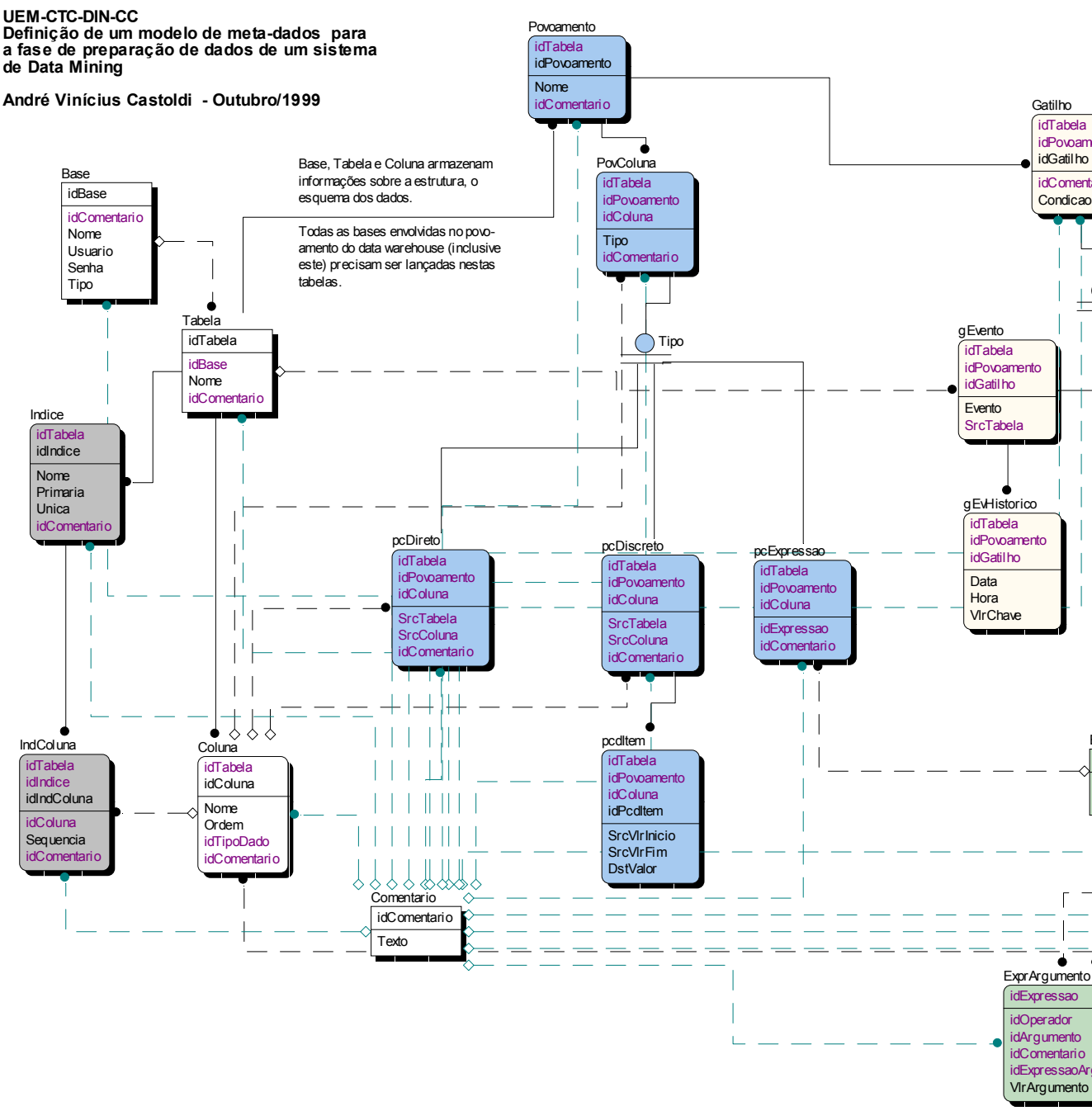


Figura 9.23: Modelo de Metadados

9.4 ESPECIFICAÇÃO DO AMBIENTE ADesC EM E-LOTOS

9.4.1 Declaração de Tipos de Dados

```
module ModDados is
  type List is
         Nil
       | Cons (any, List)
  endtype
  type TypeInt is
         integer
  endtype
 (* tipo de dados para localização do BD/DW *)
  type Typereq is
         IdBD: integer,
         NomeBD: string,
         Endereço: string
  endtype
(* identificação das tabelas que compõem o BD/DW *)
  type Typetab is
         IdBDt: integer,
         Idtabelas: List
  endtype
(* dados do BD/DW a serem selecionados pelo usuário *)
  type Typeidd is
         IdTabela: integer,
         NomeTab: string,
         Atributos: List
  endtype
(* definição das transformações nos dados a serem realizadas *)
  type Typetran is
         IdBDr: integer,
         IdTabr: integer,
         Atributo: string,
         TipoTransf: integer,
         Expressao: string
  endtype
  type Typetec is
         IdTecnica: integer,
         NomeTec: string
  endtype
  type Typeltec is
         Nil
       | Cons (string, Typeltec)
  endtype
 (* técnica, algoritmo e os parâmetros do algoritmo definidos pelo usuário *)
  type Typetpar is
         IdTecpar: integer,
         IdAlgpar: integer,
         ListPar: List
  endtype
 (* lista dos resultados obtidos *)
  type Typelres is
         ListResul: List
  endtype
```

```
 (* resultados obtidos *)
   type Typeresu is
          IdResul: integer,
          IdTecres: integer,
          IdAlgres: integer,
          Resultados: List
   endtype
(* dados para análise dos resultados *)
   type Typean is
          IdTec: integer,
          IdAlg: integer,
          DadosAn: List
   endtype
(* resultados da análise dos resultados *)
   type Typeresan is
          IdTecan: integer,
          IdAlgan: integer,
          ResulAn: List
   endtype
 (* tipos de dados do DataSet *)
   type Typedset is
          Iddset: integer,
          ListAtribs: List
   endtype
 (* dados do Banco de Dados *)
   type Typebd is
          Atrib: string,
          Valor : any
   endtype
(* tipos de dados do metadados *)
   type Typemeta is
          Idmeta: integer,
          IdBDm: integer,
          IdTabm: integer,
          Atribm: string,
          TipoTransf: integer,
          Expres: string
   endtype
(* tipo de dados para o Servidor de Agentes *)
   type Typeag is
          Parametros: List
   endtype
(* tipo de dados para o Agente Roteador *)
   type Typerot is
          IdRota: integer,
          Endereço: string
   endtype
(* dados para o Agente Povoamento *)
   type Typepovgat is
          IdGatilho: integer,
          Evento: string,
          Tempo: integer
   endtype
   type Typepovcol is
          IdDS: integer,
          IdMeta: integer,
          TipoPov: integer,
          IdTabp: integer,
          Atribp: string,
          VlrInicio: integer,
```

```
        VlrFim: integer,
        Exprep: string
endtype
type EntGerInterface is
        GerInterfEnt    (Opcao: integer)
endtype
type SaiGerInterface is
        GerInterfSai    (Mensagem: string)
endtype
type EntSupGI is
        SupGIEnt        ( )
    |   SupGIAEnt       (Sup1Ent: Typereq, Sup2Ent: Typeidd)
    |   SupGIBdEnt      (SupBDEnt: Typereq)
    |   SupGITEnt       (SupT1Ent: Typereq, SupT2Ent: Typeidd, SupT3Ent: Typetran)
    |   SupGIDsEnt      (SupDsEnt: Typedset)
    |   SupGIResEnt     (SupREnt: integer)
    |   SupGIAnEnt      (SupAnEnt: Typean)
    |   SupGIRemEnt     (SupOpEnt: TypeInt)
endtype
type SaiSupGI is
        SupGISai        ( )
    |   SupGIASai       (SupSai: Typeidd)
    |   SupGIListSai    (SupLRSai: Typelres)
    |   SupGIResSai     (SupRSai: Typeresu)
endtype
type EntSupSA is
        SupSABdEnt      (SupA1Ent: Typereq, SupA2Ent: Typetab)
    |   SupSALDsEnt     (Idset: integer)
    |   SupSAGDsEnt     (SupDSEnt: Typedset)
    |   SupSAResEnt     (SupREnt: Typeresu)
    |   SupSAMetaEnt    (SupMEnt: Typemeta)
    |   SupSAAtEnt      (SupAt1Ent: Typepovgat, SupAt2Ent: Typepovcol, SupAt3Ent: Typedset)
endtype
type SaiSupSA is
        SupSASai        (SupSai: Typeidd)
    |   SupSABdSai      (Supbd1Sai: Typetran, Supbd2Sai: Typerbd)
    |   SupSADsSai      (SupdsSai: Typedset)
    |   SupSAMetaSai    (SupMSai: Typemeta)
endtype
type EntServidor is
        ServAgTecEnt    (ServAgCEnt: Typetec)
    |   ServAgTPEnt     (ServTPEnt: Typetpar)
    |   ServAgTranEnt   (ServAgTEnt: Typetran)
    |   ServAgPovEnt    (SaGEnt: Typepovgat, SaPEnt: Typepovcol)
    |   ServAgAnEnt     (ServAgAnEnt: Typean)
    |   ServAgRemEnt    (ServOpEnt: integer)
endtype
type SaiServidor is
        ServAgSai       (ServAgSai: Typeidd)
    |   ServAgTecSai    (ServAgTSai: Typetec)
    |   ServAgTPSai     (ServAparSai: Typetpar)
    |   ServAgAnSai     (ServAanSai: Typeresan)
    |   ServAgFimSai    ( )
endtype
type EntIntEntrada is
        InterfEntrEnt ( )
endtype
type SaiIntEntrada is
        InterfEntrSai   (IntESai: Typereq)
endtype
```

```
type EntIntDados is
        InterfDadosEnt (IntD1Ent: Typereq, IntD2Ent: Typeidd)
endtype
type SaiIntDados is
        InterfDadosSai (IntD1Sai: Typereq, IntD2Sai: Typeidd)
endtype
type EntIntTransf is
        InterfTransfEnt (IntT1Ent: Typereq, IntT2Ent: Typeidd)
endtype
type SaiIntTransf is
        InterfTransfSai (IntT1Sai: Typereq, IntT2Sai: Typeidd, IntT3Sai: Typetran)
endtype
type EntIntTecnica is
        InterfTecLisEnt (IntTLEnt: Typeltec)
    |   InterfTecParEnt(IntTPEnt: Typetpar)
endtype
type SaiIntTecnica is
        InterfTecSai    (IntCSai: Typetec)
    |   InterfTecParSai(IntTPSai: Typetpar)
endtype
type EntIntResultados is
        InterfLResulEnt(IntLREnt: Typelres)
    |   InterfResulEnt  (IntREnt: Typeresu)
endtype
type SaiIntResultados is
        InterfResulSai  (IntRSai: TypeInt)
    |   InterfRFimSai   ( )
endtype
type EntIntAnalise is
        InterfAnaliseEnt(IntAEnt: Typean)
endtype
type SaiIntAnalise is
        InterfAnaliseSai( )
endtype
type EntIntPovoamento is
        InterfPovEnt    ( )
endtype
type SaiIntPovoamento is
        InterfPovSai    (IntPgSai: Typepovgat, IntPcSai: Typepovcol)
endtype
type EntBancoDados is
        BancoDadosAEnt (BdA1Ent: Typereq, BdAa2Ent: Typetab)
    |   BancoDadosBEnt (BdB1Ent: Typereq, BdB2Ent: Typeidd)
endtype
type SaiBancoDados is
        BancoDadosASai (BdA1Sai: Typereq, BdA2Sai: Typeidd)
    |   BancoDadosBSai (BdBSai: Typebd)
endtype
type EntDataSet is
        DataSetLEnt     (Idset: integer)
    |   DataSetGEnt     (DsGEnt: Typedset)
endtype
type SaiDataSet is
        DataSetLSai     (DsLSai: Typedset)
    |   DataSetGSai     ( )
endtype
type EntMetadados is
        MetadadosLEnt (IdMD: TypeInt)
    |   MetadadosGEnt (MdG1Ent: Typereq, MdG2Ent: Typeidd, MdG3Ent: Typetran)
    |   MetadadosAtEnt (MdA1Ent: Typepovgat, MdA2Ent: Typepovcol)
```

```
endtype
type SaiMetadados is
        MetadadosLSai  (MdLSai: Typemeta)
   |    MetadadosGSai  (MdG1Sai: Typereq, MdG2Sai: Typeidd)
   |    MetadadosAtSai ( )
endtype
type EntResultados is
        ResultadosEnt   ( )
   |    ResultadosLEnt (IdRe:TypeInt)
   |    ResultadosGEnt (ResGEnt: Typeresu)
endtype
type SaiResultados is
        ResultadosListSai (ResListSai: TypeIres)
   |    ResultadosLSai(ResLSai: Typeresu)
   |    ResultadosSai  ( )
endtype
type EntGerServicos is
        GerServicosEnt ( )
endtype
type SaiGerServicos is
        GerServicosSai( )
endtype
type EntCoordTransporte is
        CoordTranspEnt (CtEnt: Typereq)
endtype
type SaiCoordTransporte is
        CoordTranspSai (CtSai: Typereq)
endtype
type EntAgBusca is
        AgBuscaEnt      (AbEnt: Typereq)
endtype
type SaiAgBusca is
        AgBuscaSai      (Ab1Sai: Typereq, Ab2Sai: Typetab)
endtype
type EntAgRoteador is
        AgRoteadorEnt (ArEnt: Typereq)
endtype
type SaiAgRoteador is
        AgRoteadorSai (ArSai: Typerot)
endtype
type EntAgTransf is
        AgTransfEnt     (At1Ent: Typetran, At2Ent: Typebd)
endtype
type SaiAgTransf is
        AgTransfSai (At1Sai: Typetran, At2Sai: Typedset)
endtype
type EntAgTecnica is
        AgTecEnt        (AtecEnt: Typetec)
   |    AgTecParEnt    (AtPar1Ent: Typetpar, AtPar2Ent: Typedset)
endtype
type SaiAgTecnica is
        AgTecParSai     (AtecParSai: Typetpar)
   |    AgTecResSai     (AtecResSai: Typeresu)
endtype
type EntAgAnalise is
        AgAnaliseEnt    (AanEnt: Typean)
endtype
type SaiAgAnalise is
        AgAnaliseSai    (AanSai: Typeresan)
endtype
```

```
type EntAgPovoamento is
        AgPovamEnt    (ApcEnt: Typepovcol, Apm: Typemeta)
  endtype
  type SaiAgPovoamento is
        AgPovoamSai   (ApSai: Typedset)
  endtype
  type PortaEntGI is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntGI: EntGerInterface
  endtype
  type PortaSaiGI is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiGI: SaiGerInterface
  endtype
  type PortaEntSI is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntSI: EntSupGI
  endtype
  type PortaSaiSI is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiSI: SaiSupGI
  endtype
  type PortaEntSS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntSS: EntSupSA
  endtype
  type PortaSaiSS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiSS: SaiSupSA
  endtype
  type PortaEntSA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntSA: EntServidor
  endtype
  type PortaSaiSA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiSA: SaiServidor
  endtype
  type PortaEntIE is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIE: EntIntEntrada
  endtype
  type PortaSaiIE is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIE: SaiIntEntrada
  endtype
  type PortaEntID is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntID: EntIntDados
```

```
endtype
type PortaSaiID is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiID: SaiIntDados
endtype
type PortaEntIT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIT: EntIntTransf
endtype
type PortaSaiIT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIT: SaiIntTransf
endtype
type PortaEntIC is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIT: EntIntTecnica
endtype
type PortaSaiIC is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIC: SaiIntTecnica
endtype
type PortaEntIR is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIR: EntIntResultados
endtype
type PortaSaiIR is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIR: SaiIntResultados
endtype
type PortaEntIA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIA: EntIntAnalise
endtype
type PortaSaiIA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIA: SaiIntAnalise
endtype
type PortaEntIP is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntIP: EntIntPovoamento
endtype
type PortaSaiIP is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiIP: SaiIntPovoamento
endtype
type PortaEntGS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntGS: EntGerServicos
```

```
endtype
type PortaSaiGS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiGS: SaiGerServicos
endtype
type PortaEntCT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntCT: EntCoordTransporte
endtype
type PortaSaiCT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiCT: SaiCoordTransporte
endtype
type PortaEntAB is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntAB: EntAgBusca
endtype
type PortaSaiAB is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiAB: SaiAgBusca
endtype
type PortaEntAT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntAT: EntAgTransf
endtype
type PortaSaiAT is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiAT: SaiAgTransf
endtype
type PortaEntAC is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntAC: EntAgTecnica
endtype
type PortaSaiAC is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiAC: SaiAgTecnica
endtype
type PortaEntAA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntAA: EntAgAnalise
endtype
type PortaSaiAA is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiAA: SaiAgAnalise
endtype
type PortaEntAP is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntAP: EntAgPovoamento
```

```
  endtype
  type PortaSaiAP is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiAP: SaiAgPovoamento
  endtype
  type PortaEntBD is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntBD: EntBancoDados
  endtype
  type PortaSaiBD is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiBD: SaiBancoDados
  endtype
  type PortaEntDS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntDS: EntDataSet
  endtype
  type PortaSaiDS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiDS: SaiDataSet
  endtype
  type PortaEntMD is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntMD: EntMetadados
  endtype
  type PortaSaiMD is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiMD: SaiMetadados
  endtype
  type PortaEntRS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        EntRS: EntResultados
  endtype
  type PortaSaiRS is
        IdDestino: TypeInt,
        IdOrigem: TypeInt,
        SaiRS: SaiResultados
  endtype
endmod (* Dados *)
```

9.4.2 Declaração dos Módulos

```
module modUsuario import modDados is
  process Usuario [pent: PortaEntGI, psai: PortaSaiGI] is
      loop
          var mensagem: string,
              idest: TypeInt,
              idorig: TypeInt,
              opcao: TypeInt
          in
```

```
            ?idest := 1;
            ?idorig := 0;
            hide ent_opcao: TypeInt in
                trap
                    exit is
                        write ("Fim de Execução")
                    endexit
                in
                    loop Fim in
                        ent_opcao ( ?opcao) ;
                        case opcao is
                            1 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 2 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 3 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 4 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 5 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 6 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem))
                          | 7 -> pent (!idest, !idorig, GerInterfEnt (!opcao)) ;
                                  psai (?idest, ?idorig, GerInterfSai (?mensagem)) ;
                                  break Fim
                        endcase
                    endloop
                endtrap
            endhide
        endvar
  endproc
endmod (* Usuario *)

module modGerInterfaces import modDados is
  function CriarGI (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := "Gerente de Interfaces" ;
            true
        endvar
  endfunc
  function RemoverGI (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process GerenteInterfaces [pent: PortaEntGI, psai: PortaSaiGI, pesi: PortaEntSI,
                pssi: PortaSaiSI, pesa: PortaEntSA, pssa: PortaSaiSA, peie: PortaEntIE,
                psie: PortaSaiIE, peid: PortaEntID, psid: PortaSaiID, peit: PortaEntIT,
                psit: PortaSaiIT, peic: PortaEntIC, psic: PortaSaiIC, peir: PortaEntIR,
                psir: PortaSaiIR, peia: PortaEntIA, psia: PortaSaiIA, peip: PortaEntIP,
                psip: PortaSaiIP ] is
        var idest: TypeInt,
```

```
            idorig: TypeInt,
            opcao: TypInt,
            men: string
        in
            pent ( ?idest, ?idorig, GerInterfEnt (?opcao)) ;
            Case opcao is
                  1 -> InicializarAmbiente [psai]
              |   2 -> PrepararAmbiente [psai, pesi, pssi, pesa, pssa, peie, psie, peid, psid, peit,
                                         psit]
              |   3 -> AplicarTécnica [psai, pesi, pssi, pesa, pssa, peic, psic]
              |   4 -> VisualizarResultados [psai, pesi, pssi, pesa, pssa, peir, psir]
              |   5 -> Analisar [psai, pesi, pssi, pesa, pssa, peir, psir, peia, psia]
              |   6 -> PovoarDados [psai, pesi, pssi, pesa, pssa, peip, psip]
              |   7 -> FinalizarAmbiente [psai] (men)
            endcase
        endvar
endproc
process InicializarAmbiente [psai: PortaSaiGI] is
        var idest: TypeInt,
            idorig: TypeInt,
            cria: boolean,
            men: string
        in
            ?cria := CriarGI (1) ;
            ?cria := CriarSU (2) ;
            ?cria := CriarSA (3) ;
            ?idest : = 0 ;
            ?idorig := 1 ;
            ?men := “Ambiente Inicializado” ;
            psai  (!idest, !idorig, GerInterfSai (!men)) ;
        endvar
endproc
process PrepararAmbiente [psai: PortaSaiGI, pesi: PortaEntSI, pssi: PortaSaiSI,
                  pesa: PortaEntSA, pssa: PortaSaiSA, peie: PortaEntIE, psie: PortaSaiIE,
                  peid: PortaEntID, psid: PortaSaiID, peit: PortaEntIT, psit: PortaSaiIT] is
        var idest: TypeInt,
            idorig: TypeInt,
            cria: boolean,
            infBD: Typereq,
            atributos: Typeidd,
            inftransf: Typetran
        in
        (
            ?cria := CriarIE (4) ;
            ?cria := CriarID (5) ;
            ?cria := CriarIT (6) ;
            ?cria := CriarAgentes (1) ;
            ?cria := CriarTabelas (1) ;
            ?idorig := 1 ;
            ?idest := 4 ;
            peie  (!idest, !idorig, InterfEntrEnt ( ))
         []
            psie (?idest, ?idorig, InterfEntrSai (?infBD)) ;
            ?idorig := 1 ;
            ?idest := 2 ;
            pesi (!idest, !idorig, SupGIBdEnt (!infBD))
         []
            MostrarAtributos [pssi, peid]
         []
            psid (?idest, ?idorig, InterfDadosSai (?infBD, ?atributos)) ;
```

```
            ?idorig := 1 ;
            ?idest := 6 ;
            peit (!idest, !idorig, InterfTransfEnt (!infBD, !atributos))
        []
            psit (?idest, ?idorig, InterfTransfSai (?infBD, ?atributos, ?inftranf)) ;
            ?idorig := 1 ;
            ?idest := 2 ;
            pesi  (!idest, !idorig, SupGITEnt (!infBD, !atributos, !inftransf))
        []
            pssi (?idest, ?idorig, SupGISai ( )) ;
            FinalizarOpcao [psai, pesa, pssa] (1)
    endvar
endproc
process MostrarAtributos [pssi: PortaSaiSI, peid: PortaSaiID] is
    var idest: TypeInt,
        idorig: TypeInt,
        infBD: Typereq,
        atributos: Typeidd
    in
        pssi ( ?idest, ?idorig, SupGIASai (?infBD, ?atributos) ;
        ?idorig := 1 ;
        ?idest := 5 ;
        peid  (!idest, !idorig, InterfDadosEnt (!infBD, !atributos)
    endvar
endproc
process AplicarTécnica [psai: PortaSaiGI, pssi: PortaSaiSI, pesa: PortaEntSA,
                pssa: PortaSaiSA, peic: PortaEntIC, psic: PortaSaiIC] is
    var idest: TypeInt,
        idorig: TypeInt,
        opcao: integer,
        cria: boolean,
        listec: Typeltec,
        inftec: Typetec,
        partec: Typetpar,
        infBD: Typereq,
        atributos: Typeidd,
        inftransf: Typetran
    in
        ?cria := CriarIC (7) ;
        ?cria := CriarAgentes (2) ;
        ?cria := CriarTabelas (2) ;
        ?idorig := 1 ;
        ?idest := 7 ;
    (* Carregar lista de técnicas *)
        hide ent_listec: Typeltec in
            write (“Relacione as técnicas disponíveis”) ;
            ent_listec (?listec) ;
          (
            peic (!idest, !idorig, InterfTecListEnt (!listec))
          []
            psic (?idest, ?idorig, InterfTecSai (?inftec)) ;
            ?idorig := 1 ;
            ?idest := 3 ;
            pesa (!idest, !idorig, ServAgTecEnt (!inftec))
          []
            pssa (?idest, ?idorig, ServAgTPSai (?partec)) ;
            ?idorig := 1 ;
            ?idest := 7 ;
            peic (!idest, !idorig, InterfTecParEnt (!partec))
          []
```

```
                psic (?idest, ?idorig, InterfTecParSai (?partec)) ;
                ?idorig := 1 ;
                ?idest := 3 ;
                pesa (!idest, !idorig, ServAgTPEnt (!partec))
            []
                pssi (?idest, ?idorig, SupGISai ( )) ;
                FinalizarOpcao [psai, pesa, pssa] (2)
            )
        endhide
    endvar
endproc
process VisualizarResultados [psai: PortaSaiGI, pesi: PortaEntSI, pssi: PortaSaiSI,
            pesa: PortaEntSA, pssa: PortaSaiSA, peir: PortaEntIR, psir: PortaSaiIR] is
    var infresul: Typeresu in
        VerResultados [pesi, pssi, peir, psir] (infresul) ;
        FinalizarOpcao [psai, pesa, pssa] (3)
    endvar
endproc
process VerResultados [pesi: PortaEntSI, pssi: PortaSaiSI, peir: PortaEntIR, psir: PortaSaiIR]
            (infresul: Typeresu) is
    var idest: TypeInt,
        idorig: TypeInt,
        idresul: integer,
        cria: boolean,
        listid: TypeIres
    in
        ?cria := CriarIR (8) ;
        ?cria := CriarTabelas (3) ;
        ?idorig := 1 ;
        ?idest := 2 ;
        pesi (!idest, !idorig, SupGIEnt ( ))
     []
        pssi (?idest, ?idorig, SupGIListSai (?listid)) ;
        ?idorig := 1 ;
        ?idest := 8 ;
        peir (!idest, !idorig, InterfLResulEnt (!listid))
     []
        psir (?idest, ?idorig, InterfResulSai (?idresul)) ;
        ?idorig := 1 ;
        ?idest := 2 ;
        pesi (!idest, !idorig, SupGIResEnt (!idresul))
     []
        pssi (?idest, ?idorig, SupGIResSai (?infresul)) ;
        ?idorig := 1 ;
        ?idest := 8 ;
        peir (!idest, !idorig, InterfResulEnt (!infresul))
     []
        psir (?idest, ?idorig, InterfRFimSai ( )) ;
process Analisar [psai: PortaSaiGI, pesi: PortaEntSI, pssi: PortaSaiSI, pesa: PortaEntSA,
            pssa: PortaSaiSA, peir:PortaEntIR, psir:PortaSaiIR, peia: PortaEntIA,
            psia: PortaSaiIA] is
    var idest: TypeInt,
        idorig: TypeInt,
        cria: boolean,
        infresul: Typeresu,
        infan: Typean,
        resulan: Typeresan
    in
        ?cria := CriarIA (9) ;
        ?cria := CriarAgentes (3) ;
```

```
            ?cria := CriarTabelas (3) ;
            VerResultados [pesi, pssi, pesa, pssa, peir, psir] ;
            IdTec: IdTecres ;
            IdAlg: IdAlgres ;
            DadosAn: Resultados ;
            ?idorig := 1 ;
            ?idest := 3 ;
            pesa (!idest, !idorig, ServAgAnEnt (!infan)) ;
         []
            pssa (?idest, ?idorig, ServAgAnSai (?resulan) ;
            ?idorig := 1 ;
            ?idest := 9 ;
            peia (!idest, !idorig, InterfAnaliseEnt (!resulan))
         []
            psia (?idest, ?idorig, InterfAnaliseSai ( )) ;
            FinalizarOpcao [psai, pesa, pssa] (4)
endproc
process PovoarDados [psai: PortaSaiGI, pssi: PortaSaiGI, pesa: PortaEntSA,
                pssa: PortaSaiSA, peip: PortaEntIP, psip: PortaSaiIP] is
      var idest: TypeInt,
            idorig: TypeInt,
            cria: boolean,
            gatilho: Typepovgat,
            infpov: Typepovcol
      in
            ?cria := CriarIA (10) ;
            ?cria := CriarAgentes (4) ;
            ?cria := CriarTabelas (4) ;
            ?idorig := 1 ;
            ?idest := 10 ;
            peip (!idest, !idorig, InterfPovEnt ( )) ;
         []
            psip (?idest, ?idorig, InterfPovSai (?gatilho, ?infpov)) ;
            ?idorig := 1 ;
            ?idest := 3 ;
            pesa (!idest, !idorig, ServAgPovEnt (!gatilho, !infpov))
         []
            pssi (?idest, ?idorig, SupGISai ( )) ;
            FinalizarOpcao [psai, pesa, pssa] (5)
endproc
process FinalizarOpcao [psai: PortaSaiGI, pesa: PortaEntSA, pssa: PortaSaiSA]
                (opcao: TypeInt) is
      var Idest: TypeInt,
            Idorig: TypeInt,
            rem: boolean,
            men: string
      in
            Case opcao is
                  1 -> ?rem := RemoverIE (4) ;
                        ?rem := RemoverID (5) ;
                        ?rem := RemoverIT (6) ;
               |  2 -> ?rem := RemoverIC (7) ;
               |  3 -> ?rem := RemoverIR (8) ;
               |  4 -> ?rem := RemoverIR (8) ;
                        ?rem := RemoverIA (9) ;
               |  5 -> ?rem := RemoverIP (10) ;
            endcase
            ?idorig := 1 ;
            ?idest := 3 ;
            pesa  (!idest, !idorig, ServAgRemEnt (!opcao))
```

```
            []
               pssa (?idest, ?idorig, ServAgFimSai ( )) ;
               pesi  (!idest, !idorig, SupGIRemEnt (!opcao))
            []
               pssi (?idest, ?idorig, SupGISai ( )) ;
               ?men := “Fase Concluída” ;
               ?idorig := 1 ;
               ?idest := 0 ;
               psai (!idest, !idorig, GerInterfSai (!men))
         endvar
  endproc
  process FinalizarAmbiente [psai: PortaSaiGI, pent: PortaEntGI, psai: PortaSaiGI] is
         var idest: TypeInt,
               idorig: TypeInt,
               rem: boolean,
               men: string
         in
               ?rem := RemoverGI (1) ;
               ?rem := RemoverSU (2) ;
               ?rem := RemoverSA (3) ;
               ?men := “Ambiente Finalizado” ;
               ?idorig := 1 ;
               ?idest := 0 ;
               psai  (!idest, !idorig, GerInterfSai (!men))
         endvar
  endproc
endmod (* GerenteInterfaces *)

module modInterfEntrada import modDados is
  function CriarIE (Id: integer) : boolean is
         var idobj: integer,
               nome: string
         in
               ?idobj := Id;
               ?nome := “Interface Entrada” ;
               true
         endvar
  endfunc
  function RemoverIE (Id: integer) : boolean is
         var idobj: integer,
               nome: string
         in
               ?idobj := Id;
               ?nome := “ ” ;
               true
         endvar
  endfunc
  process InterfaceEntrada [peie: PortaEntIE, psie: PortaSaiIE] is
         EntrarInfBD [peie, psie]
  endproc
  process EntrarInfBD [peie: PortaEntIE, psie: PortaSaiIE] is
         var idest: TypeInt,
               idorig: TypeInt,
               infBD: Typereq
         in
               peie (?idest, ?idorig, InterfEntrEnt ( )) ;
               ?idorig := 4 ;
               ?idest := 1 ;
         (* Usuário entra dados sobre o Banco de Dados *)
               hide ent_infBD: Typereq in
```

```
                write (“Entrar informações sobre o banco de dados”) ;
                ent_infBD (?infBD) ;
                psie (!idest, !idorig, InterfEntrSai (!infBD))
            endhide
        endvar
  endproc
endmod (* InterfaceEntrada *)

module modInterfDados import modDados is
  function CriarID (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Interface Dados” ;
            true
        endvar
      endfunc
  function RemoverID (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process InterfaceDados [peid: PortaEntID, psid: PortaSaiID] is
        SelecionarAtributos [peid, psid]
  endproc
  process SelecionarAtributos [peid: PortaEntID, psid: PortaSaiID] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq,
            atributos: Typeidd
        in
            peid (?idest, ?idorig, InterfDadosEnt (?infBD, ?atributos)) ;
            ?idorig := 5 ;
            ?idest := 1 ;
        (* Usuário seleciona atributos *)
            hide ent_atributos: Typeidd in
                write (“Selecionar atributos da seguinte lista: ”, atributos) ;
                ent_atributos (?atributos) ;
                psid (!idest, !idorig, InterfDadosSai (!infBD!atributos))
            endhide
        endvar
  endproc
endmod (* InterfaceDados *)

module modInterfTransf import modDados is
  function CriarIT (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Interface Transformação” ;
            true
        endvar
  endfunc
  function RemoverIT (Id: integer) : boolean is
```

```
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process InterfaceTransf [peit: PortaEntIT, psit: PortaSaiIT] is
        DefinirTransformacao [peit, psit]
  endproc
  process DefinirTransformacao [peit: PortaEntIT, psit: PortaSaiIT] is
        var idest: TypeInt,
            idorig: TypeInt,
            atributos: Typeidd,
            deftran: Typetran
        in
            peit (?idest, ?idorig, InterfTransfEnt (?atributos)) ;
            ?idorig := 6 ;
            ?idest := 1 ;
        (* Usuário define transformações *)
            hide ent_deftran: Typetran in
                write ("Definir transformações sobre os seguintes atributos: ", atributos) ;
                ent_deftran (?deftran) ;
                psit (!idest, !idorig, InterfTransfSai (!deftran))
            endhide
        endvar
  endproc
endmod (* InterfaceTransf *)

module modInterfTecnica import modDados is
  function CriarIC (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := "Interface Técnica" ;
            true
        endvar
  endfunc
  function RemoverIC (Id: integer) : boolean  is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process InterfaceTecnica [peic: PortaEntIC, psic: PortaSaiIC] is
        EscolherTecnica [peic, psic]
    []  EntrarParametros [peic, psic]
  endproc
  process EscolherTecnica [peic: PortaEntIC, psic: PortaSaiIC] is
        var idest: TypeInt,
            idorig: TypeInt,
            listec: Typeltec,
            inftec: Typetec
        in
            peic (?idest, ?idorig, InterfTecListEnt (?listec))
```

```
            ?idorig := 7 ;
            ?idest := 1 ;
        (* O usuário escolhe a técnica a ser aplicada *)
            hide ent_inftec: Typetec in
                write (“Escolher uma das seguintes técnicas: ”, listec) ;
                ent_inftec (?inftec) ;
                psic (!idest, !idorig, InterfTecSai (!inftec))
            endhide
        endvar
  endproc
  process EntrarParametros [peic: PortaEntIC, psic: PortaSaiIC] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftec: Typetec,
            partec: Typetpar
        in
            peic (?idest, ?idorig, InterfTecParEnt (?partec))
            ?idorig := 7 ;
            ?idest := 1 ;
        (* O usuário define os parâmetros da técnica a ser aplicada *)
            hide ent_partec: Typetpar in
                write (“Informe parâmetros da técnica”) ;
                ent_partec (?infBD) ;
                psic (!idest, !idorig, InterfTecParSai (!partec))
            endhide
        endvar
  endproc
endmod (* InterfaceTecnica *)

module modInterfResultados import modDados is
  function CriarIR (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Interface Resultados” ;
            true
        endvar
  endfunc
  function RemoverIR (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process InterfaceResultados [peir: PortaEntIR, psir: PortaSaiIR] is
        MostrarResultados [peir, psir]
  endproc
  process MostrarResultados [peir: PortaEntIR, psir: PortaSaiIR] is
        var idest: TypeInt,
            idorig: TypeInt,
            listid: TypeIres,
            codid: TypeInt,
            infresul: Typeresu
        in
            peir (?idest, ?idorig, InterfLResulEnt (?listid)) ;
            ?idorig := 8 ;
```

```
            ?idest := 1 ;
        (* Usuário identifica a tabela de resultados a serem exibidos *)
            hide ent_codid: TypeInt in
                write (“Entrar com a identificação da tabela de resultados”) ;
                ent_codid (?codid) ;
              (
                psir (!idest, !idorig, InterfLResulSai (!codid))
              []
                peir (?idest, ?idorig, InterfResulEnt (?infresul)) ;
                ?idorig := 8 ;
                ?idest := 1 ;
                (* Os resultados são exibidos na tela *)
                write (“Resultados: “, infresul) ;
                psir (!idest, !idorig, InterfResulSai ( ))
              )
            endhide
        endvar
  endproc
endmod (* InterfaceResultados *)

module modInterfAnalise import modDados is
  function CriarIA (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Interface Análise” ;
            true
        endvar
  endfunc
  function RemoverIA (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process InterfaceAnalise [peia: PortaEntIA, psia: PortaSaiIA] is
        MostrarAnalise [peia, psia]
  endproc
  process MostrarAnálise [peia: PortaEntIA, psia: PortaSaiIA] is
        var idest: TypeInt,
            idorig: TypeInt,
            infan: Typean
        in
            peia (?idest, ?idorig, InterfAnaliseEnt (?infan)) ;
            ?idorig := 9 ;
            ?idest := 1 ;
        (* Mostrar resultados da análise na tela *)
            write (“Resultados da análise: “, infan) ;
            psia (!idest, !idorig, InterfAnaliseSai ( ))
        endvar
  endproc
endmod (* InterfaceAnalise *)

module modInterfPovoamento import modDados is
  function CriarIP (Id: integer) : boolean is
        var idobj: integer,
```

```
            nome: string
        in
            ?idobj := Id;
            ?nome := “Interface Povoamento” ;
            true
        endvar
  endfunc
  function RemoverIP (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process InterfacePovoamento [peip: PortaEntIP, psip: PortaSaiIP] is
        DefinirPovoamento [peip, psip]
  endproc
  process DefinirPovoamento [peip: PortaEntIP, psip: PortaSaiIP] is
        var idest: TypeInt,
            idorig: TypeInt,
            gatilho: Typepovgat,
            infpov: Typepovcol
        in
            peip (?idest, ?idorig, InterfPovEnt ( )) ;
            ?idorig := 10 ;
            ?idest := 1 ;
        (* Definição dos parâmetros de povoamento pelo usuário *)
            hide ent_povoam (gat: Typepovgat, povcol: Typepovcol) in
                write (“Definir os parâmetros de povoamento”) ;
                ent_povoam (?gat, ?povcol) ;
                ?gatilho := gat ;
                ?infpov := povcol ;
                psip (!idest, !idorig, InterfPovSai (!gatilho, !infpov))
            endhide
        endvar
  endproc
endmod (* InterfacePovoamento *)

module modSuporte import modDados is
  function CriarSU (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Suporte” ;
            true
        endvar
  endfunc
  function RemoverSU(Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  function CriarTabelas (Id: integer) : boolean is
```

```
        var cria: boolean
        in
            case id is
                1 -> ?cria := CriarBD (11) ;
                     ?cria := CriarMD (12) ;
                     ?cria := CriarDS (13)
              | 2 -> ?cria := CriarDS (12)
                     ?cria := CriarRS (14)
              | 3 -> ?cria := CriarRS (14)
              | 4 -> ?cria := CriarMD (12)
                     ?cria := CriarDS (13)
            endcase
            true
        endvar
endfunc
function RemoverTabelas (Id: integer) : boolean is
        var rem: boolean
        in
            case id is
                1 -> ?rem := RemoverBD (11) ;
                     ?rem := RemoverMD (12) ;
                     ?rem := RemoverDS (13)
              | 2 -> ?rem := RemoverDS (13)
                     ?rem := RemoverRS (14)
              | 3 -> ?rem := RemoverRS (14)
              | 4 -> ?rem := RemoverMD (12)
                     ?rem := RemoverDS (13)
            endcase
            true
        endvar
endfunc
process Suporte [pesi: PortaEntSI, pssi: PortaSaiSI, pess: PortaEntSS, psss: PortaSaiSS,
                 pebd: PortaEntBD, psbd: PortaSaiBD, peds: PortaEntDS, psds: PortaSaiDS,
                 pemd: PortaEntMD, psmd: PortaSaiMD, pere: PortaEntRS, psre: PortaSaiRS] is
        LocalizarBD [pesi, psss]
    []  VerAtributos [pssi, pess, pebd, psbd]
    []  GravarDefinicoes [pesi, psss, pebd, psbd, pemd, psmd]
    []  GravarDados [pesi, pess, peds, psds]
    []  CarregarDados [pess, psss, peds, psds]
    []  GravarResultados [pess, pssi, pere, psre]
    []  CarregarResultados [pesi, pssi, pere, psre]
    []  CarregarMetadados [pess, psss, pemd, psmd]
    []  AtualizarDados [pess, pssiI, pemd, psmd, peds, psds]
    []  RemoverTabs [pesi, pssi]
endproc
process LocalizarBD [pesi: PortaEntSI, psss: PortaSaiSS] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq
        in
            pesi (?idest, ?idorig, SupGIBdEnt (infBD)) ;
            ?idorig := 2 ;
            ?idest := 3 ;
        (* Verificar endereço do banco de dados *)
            psss (!idest, !idorig, SupSABdSai (infBD))
        endvar
endproc
process VerAtributos [pssi: PortaSaiSI, pess: PortaEntSS,
                      pebd: PortaEntBD, psbd: PortaSaiBD] is
        var idest: TypeInt,
```

```
            idorig: TypeInt,
            infBD: Typereq,
           inftabelas: Typetab,
           atributos: Typeidd
        in
            pess (?idest, ?idorig, SupSABdEnt (?infBD, ?inftabelas)) ;
            ?idorig := 2 ;
            ?idest := 11 ;
        (* O Suporte identifica os atributos de cada tabela *)
            pebd (!idest, !idorig, BancoDadosAEnt (!infBD, !inftabelas))
         []
            psbd (?idest, ?idorig, BancoDadosASai (?infBD?atributos)) ;
            ?idorig := 2 ;
            ?idest := 1 ;
            pssi (!idest, !idorig, SupGIASai (!infBD, !atributos))
        endvar
endproc
process GravarDefinicoes [pesi: PortaEntSI, psss: PortaSaiSS, pebd: PortaEntBD,
                psbd: PortaSaiBD, pemd: PortaEntMD, psmd: PortaSaiMD] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq
            atributos: Typeidd,
            inftransf: Typetran
        in
            pesi (?idest, ?idorig, SupGITEnt (?infBD, ?atributos, ?inftransf)) ;
            ?idorig := 2 ;
            ?idest := 12 ;
            pemd (!idest, !idorig, MetadadosGEnt (!infBD, !atributos, !inftransf))
         []
           LerDados [pss, pebd, psbd, psmd] (Inftransf)
        endvar
endproc
process LerDados [psss: PortaSaiSS, pebd: PortaEntBD, psbd: PortaSaiBD,
                psmd: PortaSaiMD] (inftransf: Typetran) is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq
            atributos: Typeidd,
            dados: Typebd
        in
            psmd (?idest, ?idorig, MetadadosSai (?infBD, ?atributos)) ;
            ?idorig := 2 ;
            ?idest := 11 ;
            pebd (!idest, !idorig, BancoDadosBEnt (!infBD, !atributos))
         []
            psbd (?idest, ?idorig, BancoDadosBSai (?dados)) ;
            ?idorig := 2 ;
            ?idest := 3 ;
            psss (!idest, !idorig, SupSABdSai (!inftransf, !dados))
        endvar
endproc
process GravarDados [pesi: PortaEntSI, pess: PortaEntSS, peds: PortaEntDS,
                psds: PortaSaiDS] is
        var idest: TypeInt,
            idorig: TypeInt,
            conjdados: Typedset
        in
            pess (?idest, ?idorig, SupSAGDsEnt (?conjdados)) ;
            ?idorig := 2 ;
```

```
            ?idest := 13 ;
            peds (!idest, !idorig, DataSetGEnt (!conjdados))
        []
            psds (?idest, ?idorig, DataSetGSai ( )) ;
            ?idorig := 2 ;
            ?idest := 1 ;
            pssi (!idest, !idorig, SupGISai ( ))
        endvar
endproc
process CarregarDados [pess: PortaEntSS, psss: PortaSaiSS, peds: PortaEntDS,
                psds: PortaSaiDS] is
        var idest: TypeInt,
            idorig: TypeInt,
            idset: integer,
            conjdados: Typedset
        in
            pess (?idest, ?idorig, SupSALdsEnt (?idset)) ;
            ?idorig := 2 ;
            ?idest := 13 ;
            peds (!idest, !idorig, DataSetLEnt (!idset))
          []
            psds (?idest, ?idorig, DataSetLSai (?conjdados)) ;
            ?idorig := 2 ;
            ?idest := 3 ;
            psss (!idest, !idorig, SupSADsSai (!conjdados))
endproc
process GravarResultados [pess: PortaEntSS, pssi: PortaSaiSI, pere: PortaEntRS,
                psre: PortaSaiRS] is
        var idest: TypeInt,
            idorig: TypeInt,
            resultados: Typeresu
        in
            pess (?idest, ?idorig, SupSAResEnt (?resultados)) ;
            ?idorig := 2 ;
            ?idest := 14 ;
            pere ( !idest, !idorig, ResultadosGEnt (!resultados))
        []
            psre (?idest, ?idorig, ResultadosSai ( )) ;
            ?idorig := 2 ;
            ?idest := 1 ;
            pssi ( !idest, !idorig, SupGISai ( ))
        endvar
endproc
process CarregarResultados [pesi: PortaEntSI, pssi: PortaSaiSI, pere: PortaEntRS,
                psre: PortaSaiRS] is
        var idest: TypeInt,
            idorig: TypeInt,
            listid: Typelres,
            idresul: TypeInt,
            infresul: Typeresu
        in
            pesi (?idest, ?idorig, SupGIEnt ( )) ;
            ?idorig := 2 ;
            ?idest := 14 ;
            pere (!idest, !idorig, ResultadosEnt ( ))
          []
            psre (?idest, ?idorig, ResultListSai (?listid)) ;
            ?idorig := 2 ;
            ?idest := 1 ;
            pssi (!idest, !idorig, SupGIListSai (!listid))
```

```
          []
             pesi (?idest, ?idorig, SupGIResEnt (?idresul)) ;
             ?idorig := 2 ;
             ?idest := 14 ;
             pere (!idest, !idorig, ResultadosLEnt (!idresul))
          []
             psre (?idest, ?idorig, ResultadosLSai (?infresul)) ;
             ?idorig := 2 ;
             ?idest := 1 ;
             pssi  (!idest, !idorig, SupGIResSai (!infresul))
       endvar
endproc
process CarregarMetadados [pess: PortaEntSS, psss: PortaSaiSS, pemd: PortaEntMD,
             psmd: PortaSaiMD] is
       var idest: TypeInt,
          idorig: TypeInt,
          idmd: TypeInt
          infmeta: Typemeta
       in
          pess (?idest, ?idorig, SupSALMEnt (?idmd)) ;
          ?idorig := 2 ;
          ?idest := 12 ;
          pemd (!idest, !idorig, MetadadosLEnt (!idmd))
          []
          psmd (?idest, ?idorig, MetadadosLSai (?infmeta)) ;
          ?idorig := 2 ;
          ?idest := 3 ;
          psss (!idest, !idorig, SupSAMetaSai (!infmeta))
endproc
process AtualizarDados [pess: PortaEntSS, pssi: PortaSaiSI, pemd: PortaEntMD,
             psmd: PortaSaiMD, peds: PortaEntDS, psds: PortaSaiDS] is
       var idest: TypeInt,
          idorig: TypeInt,
          conjdados: Typedset,
          gatilho: Typepovgat,
          infpov: Typepov
       in
          pess (?idest, ?idorig, SupSAAtEnt (?gatilho, ?infpov, ?conjdados)) ;
          ?idorig := 2 ;
          ?idest := 12 ;
          pemd (!idest, !idorig, MetadadosAtEnt (!gatilho, !infpov))
          []
          psmd (?idest, ?idorig, MetadadosAtSai ( )) ;
          ?idorig := 2 ;
          ?idest := 13 ;
          peds (!idest, !idorig, DataSetGEnt (!conjdados))
          []
          psds (?idest, ?idorig, DataSetGSai ( )) ;
          ?idorig := 2 ;
          ?idest := 1 ;
          pssi (!idest, !idorig, SupGISai ( ))
       endvar
endproc
process RemoverTabs [pesi: PortaEntSI, pssi: PortaSaiSI]
       var opcao: TypeInt ,
          retorno: boolean
       in
          pesi (?idest, ?idorig, SupGIRemEnt (?opcao)) ;
          retorno := RemoverTabelas (opcao) ;
          ?idorig := 2 ;
```

```
            ?idest := 1 ;
            pssi (!idest, !idorig, SupGISai ( ))
        endvar
  endproc
endmod (* Suporte *)

module modBancoDados import modDados is
  function CriarBD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := "Banco de Dados" ;
            true
        endvar
  endfunc
  function RemoverBD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process BancoDados [pebd: PortaEntBD, psbd: PortaSaiBD] is
        VerAtributos [pebd, psbd]
    []  LerBancoDados [pebd, psbd]
  endproc
  process VerAtributos [pebd: PortaEntBD, psbd: PortaSaiBD] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq,
            atributos: Typeidd,
        in
            pebd (?idest, ?idorig, BancoDadosAEnt (?infBD, ?inftabelas)) ;
            ?idorig := 11 ;
            ?idest := 2 ;
         (*O BD disponibiliza os atributos de cada tabela *)
            hide ent_atrib: Typebd in
                write ("Entrar os atributos das tabelas") ;
                ent_atrib (?atributos) ;
                psbd (!idest, !idorig, BancoDadosASai (!infBD, !atributos))
        endvar
  endfproc
  process LerBancoDados [pebd: PortaEntBD, psbd: PortaSaiBD] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq,
            atributos: Typeidd,
            dados: Typebd
        in
            pebd (?idest, ?idorig, BancoDadosBEnt (?infBD, ?atributos)) ;
            ?idorig := 11 ;
            ?idest := 2 ;
         (* Os valores dos atributos são lidos *)
            hide ent_dados: Typebd in
                write ("Entrar os valores dos atributos selecionados") ;
                ent_dados (?dados) ;
                psbd (!idest, !idorig, BancoDadosBSai (!dados))
```

```
            endhide
        endvar
  endproc
endmod (* BancoDados *)

module modDataSet import modDados is
  function CriarDS (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “DataSet” ;
            true
        endvar
  endfunc
  function RemoverDS (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process DataSet [peds: PortaEntDS, psds: PortaSaiDS] is
        GravarDataSet [peds, psds]
    []  LerDataSet [peds, psds]
  endproc
  process GravarDataSet [peds: PortaEntDS, psds: PortaSaiDS] is
        var idest: TypeInt,
            idorig: TypeInt,
            conjdados: Typedset
        in
            peds (?idest, ?idorig, DataSetGEnt (?conjdados)) ;
            write (“Dados gravados no DataSet”, conjdados”) ;
            ?idorig := 13 ;
            ?idest := 2 ;
            psds (!idest, !idorig, DataSetGSai ( ))
        endvar
  endproc
  process LerDataSet [peds: PortaEntDS, psds: PortaSaiDS] is
        var idest: TypeInt,
            idorig: TypeInt,
            idset: integer,
            conjdados: Typedset
        in
            peds (?idest, ?idorig, DataSetLEnt (?idset)) ;
            ?idorig :=13 ;
            ?idest := 2 ;
        (* Leitura dos dados do arquivo DataSet *)
            hide ent_conjdados: Typedset in
                write (“Entrar dados de DataSet”) ;
                ent_conjdados (?conjdados) ;
                psds (!idest, !idorig, DataSetLSai (!conjdados))
            endhide
        endvar
  endproc
endmod (* DataSet *)

module modMetadados import modDados is
```

```
  function CriarMD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Metadados” ;
            true
        endvar
  endfunc
  function RemoverMD (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process Metadados [pemd: PortaEntMD, psmd: PortaSaiMD] is
        GravarDefinições [pemd, psmd]
    []  LerMetadados [pemd, psmd]
    []  Atualizar [pemd, psmd]
  endproc
  process GravarDefinições [pemd: PortaEntMD, psmd: PortaSaiMD] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran
        in
            pemd (?idest, ?idorig, MetadadosGEnt (?infBD, ?atributos, ?inftransf)) ;
            write (“Definições de transformações gravadas no Metadados: “, inftransf) ;
            ?idorig := 12 ;
            ?idest := 2 ;
            psmd (!idest, !idorig, MetadadosGSai (!infBD, !atributos))
        endvar
  endproc
  process LerMetadados [pemd: PortaEntMD, psmd: PortaSaiMD] is
        var idest: TypeInt,
            idorig: TypeInt,
            idmd: TypeInt,
            infmeta: Typemeta
        in
            pemd (?idest, ?idorig, MetadadosLEnt (?idmd)) ;
            ?idorig := 12 ;
            ?idest := 2 ;
            psmd (!idest, !idorig, MetadadosLSai (!infmeta))
        endvar
  endproc
  process Atualizar [pemd: PortaEntMD, psmd: PortaSaiMD] is
        var idest: TypeInt,
            idorig: TypeInt,
            gatilho: Typepovgat,
            infpov: Typepovcol
        in
            pemd (?idest, ?idorig, MetadadosAtEnt (?gatilho, ?infpov)) ;
            ?idorig := 12 ;
            ?idest := 2 ;
            psmd (!idest, !idorig, MetadadosAtSai ( ))
        endvar
  endproc
endmod (* Metadados *)
```

```
module modResultados import modDados is
  function CriarRS (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := “Resultados” ;
              true
          endvar
  endfunc
  function RemoverRS (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := “ ” ;
              true
          endvar
  endfunc
  process Resultados [pere: PortaEntRS, psre: PortaSaiRS] is
          GravarResultados [pere, psre]
     []   LerResultados [pere, psre]
  endproc
  process GravarResultados [pere: PortaEntRS, psre: PortaSaiRS] is
      loop
          var idest: TypeInt,
              idorig: TypeInt,
              idresul: integer,
              cria: boolean,
              infresul: Typeresu
          in
              pere (?idest, ?idorig, ResultadosGEnt (?infresul)) ;
              write (“Resultados gravados: “, infresul) ;
              ?idorig := 14 ;
              ?idest := 2 ;
           (* Gravação dos resultados *)
              psre (!idest, !idorig, ResultadosSai ( ))
          endvar
      endloop
  endproc
  process LerResultados [pere: PortaEntRS, psre: PortaSaiRS] is
      loop
          var idest: TypeInt,
              idorig: TypeInt,
              idresul: integer,
              cria: boolean,
              listid: Typelres,
              infresul: Typeresu
          in
              pere (?idest, ?idorig, ResultadosEnt ( )) ;
              ?idorig := 14 ;
              ?idest := 2 ;
           (* Busca a lista de resultados obtidos anteriormente *)
              hide ent_listid: Typelres in
                  write (“Relacione os identificadores das tabelas de resultados”) ;
                  ent_listid (?listid) ;
                (
                  psre (!idest, !idorig, ResultadosListSai (!listid))
```

```
                []
                   pere (?idest, ?idorig, ResultadosLEnt (?idresul)) ;
                   ?idorig := 14 ;
                   ?idest := 2 ;
                   psre (!idest, !idorig, ResultadosLSai (!infresul))
                )
             endhide
          endvar
       endloop
    endproc
endmod (* Resultados *)

module modServAgentes import modDados is
  function CriarSA (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := “Servidor de Agentes” ;
              true
          endvar
  endfunc
  function RemoverSA (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := “ ” ;
              true
          endvar
  endfunc
  function CriarAgentes (Id: integer) : boolean is
          var cria: boolean
          in
              case id is
                   1 -> ?cria := CriarAB (15) ;
                        ?cria := CriarAT (16) ;
                        ?cria := CriarGS (17) ;
                        ?cria := CriarCT (18)
                 | 2 -> ?cria := CriarAC (19)
                 | 3 -> ?cria := CriarAA (20)
                 | 4 -> ?cria := CriarAP (21)
              endcase
              true
          endvar
  endfunc
  function RemoverAgentes (Id: integer) : boolean is
          var rem: boolean
          in
              case id is
                   1 -> ?rem := RemoverAB (15) ;
                        ?rem := RemoverAT (16) ;
                        ?rem := RemoverGS (17) ;
                        ?rem := RemoverCT (18)
                 | 2 -> ?rem := RemoverAC (19)
                 | 3 -> ?rem := RemoverAA (20)
                 | 4 -> ?rem := RemoverAP (21)
              endcase
              true
          endvar
```

```
endfunc

function CopiarAgente : boolean is
        (* Procedimentos para copiar agente *)
        true
endfunc
function RetrairAgente : boolean is
        (* Procedimentos para retrair agente *)
        true
endfunc
function DesativarAgente : boolean is
        (* Procedimentos para desativar agente *)
        true
endfunc
function AtivarAgente : boolean is
        (* Procedimentos para ativar agente *)
        true
endfunc
process ServidorAgentes [pesa: PortaEntSA, pssa: PortaSaiSA, pess: PortaEntSS,
                    psss: PortaSaiSS, pect: PortaEntCT, psct: PortaSaiCT, pegs: PortaEntGS,
                    psgs: PortaSaiGS, peab: PortaEntAB, psab: PortaSaiAB, peat: PortaEntAT,
                    psat: PortaSaiAT, peac: PortaEntAC, psac: PortaSaiAC, peaa: PortaEntAA,
                    psaa: PortaSaiAA] is
        LocalizarTabelas [pess, psss, pect, psct, pegs, psgs, peab, psab]
    []  TransformarDados [pesa, pssa, pess, peab, psab, peat, psat]
    []  DefinirAlgoritmo [pssa, psss, peac, psac]
    []  AplicarAlgoritmo [pesa, pess, psss, peac, psac]
    []  RealizarAnálise [pesa, pssa, peaa, psaa]
    []  AplicarPovoamento [pesa, pess, psss, peap, psap]
    []  RemoverAgs [pesa, pssa]
endproc
process LocalizarTabelas [pess: PortaEntSS, psss:PortaSaiSS, pect: PortaEntCT,
                    psct: PortaSaiCT, pegs: PortaEntGS, psgs: PortaSaiGS, peab: PortaEntAB,
                    psab: PortaSaiAB] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq,
            bdfora, retorno: boolean,
            inftabelas: Typetab
        in
            psss (?idest, ?idorig, SupSABdSai (?infBD)) ;
            hide ent_bdfora: boolean in
                write (“Banco de dados localizado em outra máquina?”) ;
                ent_bdfora (?bdfora) ;
        (* Se BD está localizado em outra máquina, então: *)
                if  bdfora then
                   retorno = CopiarAgente ;
                   ?idorig := 3 ;
                   ?idest := 18 ;
                   pect (!idest, !idorig, CoordTranspEnt (!infBD)) ;
                   psct (?idest, ?idorig, CoordTranspSai (?infBD)) ;
                   ?idorig := 3 ;
                   ?idest := 17 ;
                   pegs (!idest, !idorig, GerServicosEnt (1)) ;
                   psgs (?idest, ?idorig, GerServicosSai ( )) ;
                   ?idorig := 3 ;
                   ?idest := 15 ;
                endif
            endhide
            peab (!idest, !idorig, AgBuscaEnt (!infBD))
```

```
          []
            psab (?idest, ?idorig, AgBuscaSai (?infBD, ?inftabelas)) ;
            ?idorig := 3 ;
            ?idest := 2 ;
            pess (!idest, !idorig, SupSABdEnt (!infBD, !inftabelas))
        endvar
endproc
process TransformarDados [pesa: PortaEntSA, pssa: PortaSaiSA, pess: PortaEntSS,
                peab: PortaEntAB, psab: PortaSaiAB, peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typedset
        in
            psss (?idest, ?idorig, SupSABdSai (?inftransf, ?dados)) ;
            ?idorig := 3 ;
            ?idest := 16 ;
            peat (!idest, !idorig, AgTransfEnt (!inftransf, !dados))
          []
            psat (?idest, ?idorig, AgTransfSai (?dset)) ;
            ?idorig := 3 ;
            ?idest := 2 ;
            pess (!idest, !idorig, SupSAGDsEnt (!dset))
        endvar
endproc
process DefinirAlgoritmo [pssa: PortaEntSA, psss: PortaSaiSS, peac: PortaEntAC,
                psac: PortaSaiAC] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftec: Typetec,
            partec: Typetpar
        in
            pesa (?idest, ?idorig, ServAgTecEnt (?inftec)) ;
            ?idorig := 3 ;
            ?idest := 19 ;
            peac (!idest, !idorig, AgTecEnt (!inftec))
          []
            psac (?idest, ?idorig, AgTecParSai (?partec)) ;
            ?idorig := 3 ;
            ?idest := 1 ;
            pssa (!idest, !idorig, ServAgTPSai (!partec))
        endvar
endproc
process AplicarAlgoritmo [pesa: PortaEntSA, pess: PortaEntSS, psss: PortaSaiSS,
                peac: PortaEntAC, psac: PortaSaiAC is
        var idest: TypeInt,
            idorig: TypeInt,
            opcao: integer,
            partec: Typetpar,
            idset: List,
            conjdados: Typedset,
            infresu: Typeresu
        in
           (
            pesa (?idest, ?idorig, ServAgTPEnt (?partec)) ;
            ?idorig := 3 ;
            ?idest := 2 ;
        (* Identificar as tabelas de DataSet *)
            hide ent_idset: List in
```

```
                write ("Identificar a tabela de DataSet") ;
                ent_idset (?idset)
            endhide
            pess (!idest, !idorig, SupSALdsEnt (!idset))
          )
          []
            psss (?idest, ?idorig, SupSADsEnt (?conjdados)) ;
            ?idorig := 3 ;
            ?idest := 19 ;
            peac (!idest, !idorig, AgTecParEnt (!partec, !conjdados))
          []
            psac (?idest, ?idorig, AgTecResSai (?infresu)) ;
            ?idorig := 3 ;
            ?idest := 2 ;
            pess (!idest, !idorig, SupSAResEnt (!infresu))
        endvar
endproc
process RealizarAnálise [pesa: PortaEntSA, pssa: PortaSaiSA, peaa: PortaEntAA,
                psaa: PortaSaiAA] is
        var idest: TypeInt,
            idorig: TypeInt,
            infan: Typean,
            resulan: Typeresan
        in
            pesa (?idest, ?idorig, ServAgAnEnt (?infan)) ;
            ?idorig := 3 ;
            ?idest := 20 ;
            peaa (!idest, !idorig, AgAnaliseEnt (!infan))
          []
            psaa (?idest, ?idorig, AgAnaliseSai (?resulan)) ;
            ?idorig := 3 ;
            ?idest := 1 ;
            pssa (!idest, !idorig, ServAgAnSai (!resulan))
       endvar
endproc
process AplicarPovoamento [pesa: PortaEntSA, pess: PortaEntSS, psss: PortaSaiSS,
                peap: PortaEntAP, psap: PortaSaiAP] is
        var idest: TypeInt,
            idorig: TypeInt,
            gatilho: Typepovgat,
            infpov: Typepovcol,
            conjmeta: Typemeta,
            dset: Typedset,
            disparar : Boolean
        in
            pesa (?idest, ?idorig, ServAgPovEnt (?gatilho, ?infpov)) ;
            ?idorig := 3 ;
            ?idest := 2 ;
            case IdGatilho is
                1 -> wait (tempo) ;
                     disparar := true
              | 2 -> disparar := true
              | 3 -> disparar := falce
            endcase
            pess (!idest, !idorig, SupSALDsEnt (!idDS))
          []
            psss (?idest, ?idorig, SupSADsSai (?conjmeta)) ;
            ?idorig := 3 ;
            ?idest := 21 ;
            peap (!idest, !idorig, AgPovoamEnt (!infpov, !conjmeta))
```

```
            []
              psap (?idest, ?idorig, AgPovoamSai (?dset)) ;
              ?idorig := 3 ;
              ?idest := 2 ;
              pess (!idest, !idorig, SupSAAtEnt (!gatilho, !infpov, !dset))
          endvar
  endproc
  process RemoverAgs [pesa: PortaEntSA, pssa: PortaSaiSA]
          var opcao: TypeInt ,
              retorno: boolean
          in
              pesa (?idest, ?idorig, ServAgRemEnt (?opcao)) ;
              retorno := RemoverAgentes (opcao) ;
              ?idorig := 2 ;
              ?idest := 1 ;
              pssa (!idest, !idorig, ServAgFimSai ( ))
          endvar
  endproc

endmod (* ServidorAgentes *)

module modAgBusca import modDados is
  function CriarAB (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := "Agente Busca" ;
              true
          endvar
  endfunc
  function RemoverAB (Id: integer) : boolean is
          var idobj: integer,
              nome: string
          in
              ?idobj := Id;
              ?nome := " " ;
              true
          endvar
  endfunc
  process AgenteBusca [peab: PortaEntAB, psab: PortaSaiAB] is
          BuscarAtributos [peab, psab]
     []   BuscarDados [peab, psab]
  endproc
  process BuscarAtributos [peab: PortaEntAB, psab: PortaSaiAB] is
          var idest: TypeInt,
              idorig: TypeInt,
              infBD: Typereq,
              inftabelas: Typetab,
          in
              peab (?idest, ?idorig, AgBuscaEnt (?infBD)) ;
              ?idorig := 15 ;
              ?idest := 3 ;
          (* Busca as tabelas do banco de dados *)
              hide ent_tab: Typeidd in
                  ("Relacionar as tabelas") ;
                  ent_tab (?inftabelas) ;
                  psab (!idest, !idorig, AgBuscaSai (!infBD, !inftabelas))
              endhide
          endvar
```

```
  endproc
endmod (* AgenteBusca *)
module modAgTransformacao import modDados is
  function CriarAT (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Agente Transformação” ;
            true
        endvar
  endfunc
  function RemoverAT (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  process AgenteTransf [peat: PortaEntAT, psat: PortaSaiAT] is
        TransformarDados [peat, psat]
  endproc
  process TransformarDados [peat: PortaEntAT, psat: PortaSaiAT] is
        var idest: TypeInt,
            idorig: TypeInt,
            inftransf: Typetran,
            dados: Typebd,
            dset: Typebd
        in
            peat (?idest, ?idorig, AgTransfEnt (?inftransf, ?dados)) ;
            ?idorig := 15 ;
            ?idest := 3 ;
         (* Transformar os dados e gerar um conjunto de dados *)
            hide ent_dset: Typebd in
                write (“Entrar com os resultados das transformações”) ;
                ent_dset (?dset) ;
                psat (!idest, !idorig, AgTransfSai (!inftransf, !dset))
            endhide
        endvar
   endproc
endmod (* AgenteTransf *)

module modAgRoteador import modDados is
  function CriarAR (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Agente Roteador” ;
            true
        endvar
  endfunc
  function RemoverAR (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
```

```
            true
        endvar
  endfunc
  process AgenteRoteador [pear: PortaEntAR, psar: PortaSaiAR] is
        DefinirRota [pear, psar]
  endproc
  process DefinirRota [pear: PortaEntAR, psar: PortaSaiAR] is
        (* Definir a rota do agente *)
  endproc
endmod (* AgenteRoteador *)

module modAgTecnica import modDados is
  function CriarAC (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := "Agente Técnica" ;
            true
        endvar
  endfunc
  function RemoverAC (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process AgenteTecnica [peac: PortaEntAC, psac: PortaSaiAC] is
        DefinirAlgoritmo [peac, psac]
    []  AplicarAlg [peac, psac]
  endproc
  process DefinirAlgoritmo [peac: PortaEntAC, psac: PortaSaiAC] is
        var idest: TypeInt,
            idorig: TypeInt,
            opcao: integer,
            inftec: Typetec,
            partec: Typetpar,
        in
            peac (?idest, ?idorig, AgTecEnt (?inftec)) ;
            ?idorig := 19 ;
            ?idest := 3 ;
            hide ent_partec: Typetpar in
                write ("Definir quais são os parâmetros para a técnica escolhida") ;
                ent_partec (?partec) ;
                psac (!idest, !idorig, AgTecParSai (!partec))
            endhide
        endvar
  endproc
  process AplicarAlg [peac: PortaEntAC, psac: PortaSaiAC] is
        var idest: TypeInt,
            idorig: TypeInt,
            opcao: integer,
            partec: Typetpar,
            dset: Typedset,
            infresu: Typeresu
        in
            peac (?idest, ?idorig, AgTecParEnt (?partec, ?dset))
```

```
                ?idorig := 19 ;
                ?idest := 3 ;
            (* Aplicar a técnica escolhida *)
                hide ent_infresu: Typeresu in
                    write (“Informar os resultados obtidos com a aplicação da técnica”) ;
                    ent_infresu (?infresu) ;
                    psac (!idest, !idorig, AgTecResSai (!infresu))
                endhide
            endvar
    endproc
endmod (* AgenteTecnica *)

module modAgAnalise import modDados is
    function CriarAA (Id: integer) : boolean is
            var idobj: integer,
                nome: string
            in
                ?idobj := Id;
                ?nome := “Agente Análise” ;
                true
            endvar
    endfunc
    function RemoverAA (Id: integer) : boolean is
            var idobj: integer,
                nome: string
            in
                ?idobj := Id;
                ?nome := “ ” ;
                true
            endvar
    endfunc
    process AgenteAnalise [peaa: PortaEntAA, psaa: PortaSaiAA] is
            RealizarAnálise [peaa, psaa]
    endproc
    process RealizarAnálise [peaa: PortaEntAA, psaa: PortaSaiAA] is
            var idest: TypeInt,
                idorig: TypeInt,
                infan: Typean,
                resulan: Typeresan
            in
                peaa (?idest, ?idorig, AgAnaliseEnt (?infan))
                ?idorig := 20 ;
                ?idest := 3 ;
            (* Faz a análise dos resultados obtidos *)
                hide ent_resulan: Typeresan in
                    write (“Informar os resultados da análise”) ;
                    ent_resulan (?resulan) ;
                    psaa (!idest, !idorig, AgAnaliseSai (!resulan))
                endhide
            endvar
    endproc
endmod (* AgenteAnalise *)

module modAgPovoamento import modDados is
    function CriarAP (Id: integer) : boolean is
            var idobj: integer,
                nome: string
            in
                ?idobj := Id;
                ?nome := “Agente Povoamento” ;
```

```
            true
        endvar
  endfunc
  function RemoverAP (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  process AgentePovoamento [peap: PortaEntAP, psap: PortaSaiAP] is
        PovoarDados [peap, psap]
  endproc
  process PovoarDados [peap: PortaEnetAP, psap: PortaSaiAP] is
        var idest: TypeInt,
            idorig: TypeInt,
            infpov: Typepovcol,
            conjmeta: Typemeta,
            dset: Typebd
        in
            peap (?idest, ?idorig, AgPovoamEnt (?infpov, ?conjmeta)) ;
            ?idorig := 21 ;
            ?idest := 3 ;
            hide ent_dset: Typebd in
                write ("Informar os resultados do povoamento") ;
                ent_dset (?dset) ;
                psap (!idest, !idorig, AgPovoamSai (!dset))
            endhide
        endvar
  endproc
endmod (* AgentePovoamento *)

module modGerServiços import modDados is
  function CriarGS (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := "Gerente de Serviços" ;
            true
  endfunc
  function RemoverGS (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := " " ;
            true
        endvar
  endfunc
  function EscolherServiço : TypeInt is
            1
        |   2
        |   3
        |   4
        |   5
        |   6
        |   7
```

```
        |  8
        |  9
  endfunc
  function AtivarServiço : boolean is
        true
  endfunc
  function NomearAgente : boolean is
        true
  endfunc
  function ControlarMigração : boolean is
        true
  endfunc
  function AdquirirDados : boolean is
        true
  endproc
  function DeterminarRota : boolean is
        true
  endfunc
  function ControlarComunicação : boolean is
        true
  endfunc
  function TratarEspera : boolean is
        true
  endfunc
  function TratarPersistência : boolean is
        true
  endfunc
  function GarantirSegurança : boolean is
        true
  endfunc
  function EvitarFalhas : boolean is
        true
  endfunc
  process GerenteServicos [pegs: PortaEntGS; psgs: PortaSaiGS] is
        var idest: TypeInt,
            idorig: TypeInt,
            idserv: TypeInt,
            retorno: boolean
        in
            pegs (?idest, ?idorig, GerServicosEnt ( )) ;
            ?Idserv = EscolherServico ;
            Case Idserv is
                    1 -> retorno = AtivarServico
                |   2 -> retorno = NomearAgente
                |   3 -> retorno = ControlarMigracao
                |   4 -> retorno = AdquirirDados
                |   5 -> retorno = ControlarComunicação
                |   6 -> retorno = TratarEspera
                |   7 -> retorno = TratarPersistencia
                |   8 -> retorno = GarantirSegurança
                |   9 -> retorno = EvitarFalhas
            endcase
            ?idorig := 17 ;
            ?idest := 3 ;
            psgs (!idest, !idorig, GerServicosSai ( ))
        endvar
  endproc
endmod (* GerenteServiços *)
```

```
module modCoordTransporte import modDados is
  function CriarCT (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “Coordenador Transporte” ;
            true
        endvar
  endfunc
  function RemoverCT (Id: integer) : boolean is
        var idobj: integer,
            nome: string
        in
            ?idobj := Id;
            ?nome := “ ” ;
            true
        endvar
  endfunc
  function ExpedirAgente : boolean is
        (* Procedimentos para expedir agente *)
        true
  endfunc
  process CoordenadorTransporte [pect: PortaEntCT, psct: PortaSaiCT] is
        CoordenarTransporte [pect, psct]
  endproc
  process CoordenarTransporte [pect: PortaEntCT, psct: PortaSaiCT] is
        var idest: TypeInt,
            idorig: TypeInt,
            infBD: Typereq,
        in
            pect (?idest, ?idorig, CoordTranspEnt (?infBD)) ;
            ?idorig := 18 ;
            ?idest := 3 ;
         (* Coordenar o transporte do agente para outra máquina *)
            psct (!idest, !idorig, CoordTranspSai (!infBD))
  endproc
endmod (* CoordenadorTransporte *)
```

9.4.3 Especificação Geral

```
specification AMBIENTE import modDados, modUsuario, modGerInterfaces, modInterfEntrada,
        modInterfDados, modInterfTransformacao, modInterfTecnica, modInterfResultados,
        modInterfAnalise, modInterfPovoamento, modSuporte,  modBancoDados, modDataSet,
        modMetadados, modResultados modServAgentes, modGerServicos,
        modCoordTransporte, modAgBusca, modAgRoteador, modAgTransf, modAgTecnica,
        modAgAnalise, modAgPovoamento is
        gates   pent: PortaEntGI, psai: PortaSaiGI, pesi: PortaEntSI, pssi: PortaSaiSI,
                pess: PortaEntSS, psss: PortaSaiSS, pesa: PortaEntSA, pssa: PortaSaiSA,
                peie: PortaEntIE, psie: PortaSaiIE, peid: PortaEntID,  psid: PortaSaiID,
                peit: PortaEntIT,  psit: PortaSaiIT, peic: PortaEntIC,  psic: PortaSaiIC,
                peir: PortaEntIR,  psir: PortaSaiIR, peia: PortaEntIA, psia: PortaSaiIA,
                peip: PortaEntIP, psip: PortaSaiIP, pegs: PortaEntGS, psgs: PortaSaiGS,
                pect: PortaEntCT, psct: PortaSaiCT, peab: PortaEntAB, psab: PortaSaiAB,
                peat: PortaEntAT, psat: PortaSaiAT, peac: PortaEntAC, psac: PortaSaiAC,
                peaa: PortaEntAA, psaa: PortaSaiAA, peap: PortaEntAP, psap: PortaSaiAP,
```

```
                pebd: PortaEntBD, psbd: PortaSaiBD, peds: PortaEntDS, psds: PortaSaiDS,
                pemd: PortaEntMT, psmd: PortaSaiMD, pere: PortaEntRS, psre: PortaSaiRS
behaviour
(
   Usuario [peus, psus, pent, psai]
   |[pent, psai]|
   (
        GerenteInterfaces [pent, psai, pesi, pssi, pesa, pssa, peie, psie, peid, psid, peit, psit,
                           peic, psic, peir, psir, peia,  psia, peip, psip]
        |[peie, psie, peid, psid, peit, psit, peic, psic, peir, peia]|
        (
                InterfaceEntrada [peie, psie]
                |||
                InterfaceDados [peid, psid]
                |||
                InterfaceTransformacao [peit, psit]
                ||
                InterfaceTecnica [peic, psic]
                |||
                InterfaceResultados [peir, psir]
                |||
                InterfaceAnalise [peia, psia]
                |||
                InterfacePovoamento [peip, psip]
        )
   )
)
|[pesa, pssa, pesi, pssi]|
(
   (
        Suporte [pesi, pssi, pess, psss, pebd, psbd, peds, psds, pemd, psme, pere, psre, peag,
                 psag]
        |[pebd, psbd, peds, psds, pemd, psme, pere, psre]|
        (
                BancoDados [pebd, psbd]
                |||
                DataSet [peds, psds]
                |||
                Metadados [pemd, psmd]
                |||
                Resultados [pere, psre]
        )
   )
   |[pess, psss]|
   (
        ServidorAgentes [pesa, pssa, pess, psss, pegs, psgs, pect, peab, psab, pear, psar,peat,
                         psat, peac, psac, peaa, psaa, peap, psap]
        |[pegs, psgs, pect, psct, peab, psab, pear, psar, peat, psat, peac, psac, peaa, psaa,
          peap]|
        (
                GerenteServicos [pegs, psgs]
                |||
                CoordenadorTransporte [pect]
                |||
                AgenteBusca [peab, psab]
                |||
                AgenteRoteador [pear, psar]
                |||
                AgenteTransformacao [peat, psat]
                |||
```

```
                AgenteTecnica [peac, psac]
                |||
                AgenteAnalise [peaa, psaa]
                |||
                AgentePovoamento [peap]
        )
    )
)
endspec
```