UNIVERSIDADE FEDERAL DE PERNAMBUCO
Centro de Ciências Exatas e da Natureza
Departamento de Informática

Pós-Graduação em Ciência da Computação

ONTOSITTER: SISTEMA DE SOLUÇÃO DE PROBLEMAS DE SAGBD BASEADO EM ONTOLOGIA

Ricardo Alexandre Afonso

Dissertação de Mestrado

RECIFE, AGOSTO/2008

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RICARDO ALEXANDRE AFONSO

ONTOSITTER: SISTEMA DE SOLUÇÃO DE PROBLEMAS DE SAGBD BASEADO EM ONTOLOGIA

DISSERTAÇÃO APRESENTADA À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: ANA CAROLINA SALGADO
CO-ORIENTADORA: PATRICIA TEDESCO

RECIFE, AGOSTO/2008

Afonso, Ricardo Alexandre

Ontositter: sistema de solução de problemas de SAGBD baseado em ontologia / Ricardo Alexandre Afonso. – Recife: O Autor, 2008.

xi, 120 folhas : il., fig., quadros.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2008.

Inclui bibliografia, glossário e anexo.

1. Banco de dados. 2. Sistemas autônomos. 3. Ontologia. 4. Sistemas multi agentes. I. Título.

025.04 CDD (22.ed.) MEI2008-111

Agradecimentos

À minha família, que composta por gente humilde, trabalhadora e mesmo distante se fez presente pelos dons que me concedeu e pelo amor que dedicou a minha criação. (Agora nossa família atinge um degrau importante na educação.)

Ao meu anjo, Gabriel, que um dia há de compreender (e me perdoar) pelas razões que me afastaram do seu convívio diário para atingir um sonho numa terra distante do toque de suas mãozinhas. (Agora teu papai vai realizar nossos sonhos.)

Aos meus amigos quase irmãos, cujas palavras nunca serão suficientes para dimensionar quão importantes foram em minha vida: Joel da Silva, Alexandre Álvaro, Vinicius Cardoso, Eduardo Almeida, Cinthya Roberta, Leobino Sampaio, Clarisse Ferrão, Claudete Werner, Val Fontanette, Darley Peres e outros tantos que cultivei nesses anos de estudo. (Agora carrego comigo o mais valioso grupo de amigos.)

Aos professores que se fizeram presentes no inicio desta jornada: Magdala Novaes, Robson Fidalgo e Eduardo Carneiro. (Agora retribuo a atenção.)

À minha orientadora Ana Carolina Salgado por me conduzir nesse processo onde muitas vezes um pequeno sinal foi suficiente para mostrar o rumo a seguir. (Agora é hora de provar que aprendi.)

À minha co-orientadora Patrícia Tedesco, por me ensinar que em uma jornada sempre há mais um caminho a ser explorado. (Agora vejo as possibilidades.)

Ao povo nordestino, que literalmente norteou minha vida e me tornou enriquecido e enriquecedor. (Agora é hora de trabalhar!)

À Luciana Campos que fez desse árido coração paulista um “coração donzelo e abestalhado!” (Agora quero mantê-lo assim pra sempre!)

Ao amigo que sempre acreditou: Ricardo Argenton Ramos. (A melhor hora de lhe agradecer foi quando me chamou pra vir, e a segunda melhor hora é agora: Obrigado!!)

À Deus! Por ter colocado cada uma dessas pessoas no meu caminho!

Resumo

Atualmente os Sistemas Gerenciadores de Bancos de Dados (SGBD) possuem grande capacidade de armazenar desde os dados obtidos pelos sistemas transacionais até as informações sintetizadas pelos sistemas de tomada de decisão. Nesse contexto, tanto a busca pela informação quanto as rotinas de manutenção deste ambiente requerem um esforço maior por parte dos administradores de bancos de dados (ABD) e a criação de rotinas e mecanismos que automatizem as tarefas de administração e gerenciamento. A aplicação dos conceitos de auto-configuração, auto-otimização, auto-recuperação e auto-proteção, utilizados pela área de computação autônoma, passaram a ser aplicados aos SGBD dando origem a um novo conceito denominado Sistemas Autônomos de Gerenciamento de Bancos de Dados (SAGBD) permitindo automatizar tarefas de detecção e correção de falhas que até então dependiam de interação humana intensa por parte dos ABD. Este trabalho tem o objetivo de aprimorar os mecanismos de identificação e solução de erros no domínio de Sistemas Autônomos de Gerenciamento de Banco de Dados. Para isso propõe-se conceitualizar o domínio de problemas em SGBD e das tarefas do ABD, formalizar os modelos conceituais destas ontologias e implementar rotinas de comunicação entre agentes de software que atuem nesse ambiente. A criação do sistema que utiliza agentes de software e ontologias recebeu o nome de OntoSitter,

Palavras chave: SGBD, SAGBD, Ontologia, Sistemas Multi Agentes.

Abstract

Currently the Database Management System (DBMS) possess great capacity to store since the data gotten for the systems do business until the information synthecized for the systems of decision taking. In this context, as much the search for the information how much the routines of maintenance of this environment require a bigger effort on the part of the database administrators (DBA) and the creation of routines and mechanisms that automatize the tasks of administration and management. The application of the concepts of self-configuration, self-optimization, self-recovery and self-protection, used for the area of autonomic computing, had passed to be applied to the DBMS giving origin to a new called concept: Autonomic Database Management System (ADBMS) allowing to automatize tasks of detection and correction of fails that until then depended on interaction intense human being on the part of the DBA. This work has the objective to improve the mechanisms of identification and solution of errors in the domain of ADBMS. For this it is considered to conceptualize the domain of problems in ADBMS, to formalize a conceptual model of this domain through the creation of a ontology and to implement a formal model of communication between software agents who act in this environment. The implementation of the mechanism that uses a domain ontology of problems found in SAGBDs received the name of OntoSitter.

Keywords: DBMS, ADBMS, Ontology, Multi Agent Systems.

Índice

1 – Introdução..... 01
1.1. Motivação..... 02
1.2. Objetivos..... 03
1.3. Contribuições..... 04
1.4. Metodologia..... 05
1.5. Organização do documento..... 05
2 - Computação Autônoma e Ontologias..... 07
2.1. Introdução..... 08
2.2. Computação autônoma..... 08
2.3. Computação Autônoma em SGBD..... 11
2.3.1. Gerenciamento e automatização de falhas em sistemas..... 12
2.3.2. Arquiteturas de implementação de sistemas multiagentes..... 14
2.3.3. Trabalhos correlatos de implementação de SA..... 15
2.4. Ontologias..... 26
2.4.1. Ontologias e SGBD..... 27
2.5. Considerações..... 29
3 – Metodologias e padrões adotados..... 31
3.1. Introdução..... 32
3.2. Methontology..... 32
3.2.1. Etapa 1: Definição de escopo..... 34
3.2.2. Etapa 2: Conceitualização..... 37
3.2.3. Etapa 3: Formalização 37
3.2.4. Etapa 4: Implementação..... 39
3.3. SADAAM (*Software Agent Development An Agile Methodology)*..... 42
3.3.1. Projeto..... 42
3.3.2. Implementação orientada a testes..... 46
3.4. Padrão de comunicação entre agentes..... 48
3.5. Considerações..... 51
4 – Definição da ontologia de domínio para SAGBD..... 54
4.1. Introdução..... 55
4.2. Definição da ontologia de domínio..... 55
4.2.1. Especificação..... 55
4.2.2. Conceituação de domínio de um SAGBD..... 56
4.2.3. Formalização das classes do OntoSitter..... 59
4.2.4. Integração de ontologias e classes..... 64
4.2.5. Implementação da ontologia em OWL..... 66
4.3. Considerações..... 70
5 – Implementações: OntoSitter e SGBD Oracle..... 72
5.1. Desenvolvimento do sistema OntoSitter..... 73
5.1.1. Projeto..... 74
5.1.2. Implementação orientada a testes..... 80
5.2. Ambiente de desenvolvimento..... 82
5.3. Implementações..... 84
5.3.1. Implementação 1: Interação agentes – ABD..... 84
5.3.2. Implementação 2: Auto-proteção e auto recuperação do SGBD..... 93
5.4. Considerações..... 99
6 – Conclusões e trabalhos Futuros..... 99
Referências..... 104
Anexo A..... 115

Lista de Figuras

Figura 2-1: Camada entre o banco de dados e as aplicações.....12
Figura 2-2: Principais causas de falhas em sistemas [Telford, 2008].....13
Figura 2-3: Arquiteturas de implementação de interação entre agentes e SGBD.....14
Figura 2-4: Mecanismo de monitoramento de objetos do IBMDB2.....24
Figura 2-5: Relatórios de performance do MSSQL2008 [MSSQL Server, 2008].....25
Figura 2-6: Objetos que representam ou não ontologia. [McGuiness et al, 2000].....26
Figura 2-7: Engenharia reversa para ontologias de BD relacionais [O´Brien 2000]....28
Figura 3-1: Hierarquia de classes.....37
Figura 3-2: Relacionamentos entre as classes.....38
Figura 3-3: Axioma de associacao entre as classes da ontologia.....38
Figura 3-4: Instancias de uma ontologia.....39
Figura 3-5: Arquitetura da metodologia SADAAM.....42
Figura 3-6: Etapas da fase de Design.....43
Figura 3-7: Diagrama de comportamento do sistema.....44
Figura 3-8: Diagrama de unidade organizacional [Clynch, 2007].....44
Figura 3-9: Diagrama de interação dos agentes.....45
Figura 3-10: Diagrama de atividades.....46
Figura 3-11: Estrutura da fase de implementação baseada em testes.....47
Figura 3-12: Categorias de classificação de agentes por Nwana [Nwana, 1996].....49
Figura 3-13: Agentes podem utilizar um facilitador.....50
Figura 4-1: Contexto para um SAGBD.....55
Figura 4-2: Hierarquia de classes.....56
Figura 4-3: Relacionamentos hierárquicos e não hierárquicos.....57
Figura 4-4: Representacao grafica de problemas e erros.....58
Figura 4-5: Exemplo de um problema em SGBD.....59
Figura 4-6: Classe Falha.....60
Figura 4-7: Classe Dicionario.....61
Figura 4-8: Classe Tarefa.....62
Figura 4-9: Classe Acao.....63
Figura 4-10: Classes necessárias ao gerenciador de agentes JADE.....64
Figura 4-11: Integracao de classes do Jade com a ontologia de domínio e SGBD.....64
Figura 4-12a: Apenas hierárquica.....66
Figura 4-12b: Hierarquica e não hierarquica.....66
Figura 4-13: Implementacao de indivíduos ou instancias de classes.....67
Figura 4-14: Propriedades funcionais e transitivas da ontologia.....67
Figura 4-15: Restriçoes da classe fata_de_comunicacao_com_o_servidor.....68
Figura 4-16: Restricoes e relacionamentos da ontologia.....69
Figura 5-1: Arquitetura do OntoSitter.....73
Figura 5-2: Diagrama de comportamento do OntoSitter.....74
Figura 5-3a: Comportamento *leLogEventos*.....74
Figura 5-3b: Comportamento *executaAcao*.....74
Figura 5-4: Diagrama de unidade organizacional do OntoSitter.....75
Figura 5-5: Sub-nivel do comportamento dos agentes.....75
Figura 5-6: Etapas do Sub-nível do comportamento dos agentes [Carneiro, 2004].....76
Figura 5-7: Hierarquia de classes e agentes.....77
Figura 5-8: Modelo de interação entre agentes do OntoSitter.....78
Figura 5-9: Modelo de atividades dos agentes OntoSitter.....80
Figura 5-10: Agentes taaSensor(n) são instanciados.....81
Figura 5-11: Diagrama de processos para criação das implementações.....83
Figura 5-12: SGBD Oracle e o armazenamento de LOG desativado.....84
Figura 5-13: Diagrama de atividades da implementacao 1.....87

Figura 5-14: Classes necessárias ao framework JADE.....88
Figura 5-15: Caminho onde o log recebe os eventos do SGBD Oracle.....89
Figura 5-16: Interface do ABD.....92
Figura 5-17: Diagrama de atividades da implementação 2.....95
Figura 5-18: Hierarquia de classes para verificação de parâmetros.....96
Figura 5-19: Classe D_Parametros e o indivíduo LOCK_SGA.....96
Figura 5-20: Propriedades do individuo **OracleServiceXE**.....97

Lista de Quadros

Quadro 2-1: Lista de definições para computação autônoma.....09
Quadro 2-2: Níveis de automação dos sistemas computacionais.....10
Quadro 2-3: Trabalhos recentes relacionados a auto-configuração em SGBD.....16
Quadro 2-4: Trabalhos recentes relacionados a auto-otimização em SGBD.....17
Quadro 2-5: Trabalhos recentes relacionados a auto-recuperação em SGBD.....18
Quadro 2-6: Trabalhos recentes relacionados a auto-proteção em SGBD.....19
Quadro 2-7: Características e aspectos da computação autônoma.....20
Quadro 2-8: Computação autônoma aplicada a SGBD.....21
Quadro 2-9: Comando SQL que realiza a coleta de informações.....22
Quadro 2-10: Recomendação de incremento de memória.....22
Quadro 2-11: Execução de comando SQL para gerar relatório de estatísticas.....23
Quadro 3-1: Comparação de metodologias.....33
Quadro 3-2: Linguagens para Ontologias.....40
Quadro 3-3: Agente 1 envia mensagem ao Agente 2.....51
Quadro 3-4: Agente 2 responde ao Agente 1.....51
Quadro 5-1: Exemplo de evento gerado pelo SGBD.....85
Quadro 5-2: Instancias e definição de papéis.....86
Quadro 5-3: Relação de protocolos.....90
Quadro 5-4: Fluxo de comunicação.....90
Quadro 5-5: Ontologia OWL contendo o individuo ORA-00018.....91
Quadro 5-6: Instancias e definição de papéis da implementação 2.....94

Glossário

AAUT	*Application Agents Under Test*
AC	Aquisição de conhecimento
ABD	Administrador de bancos de dados
ADBMS	*Autonomic database management system*
AUML	*Agents Unified Model Language*
DAMA	*Data Management Association*
DAML	*DARPA Agent Markup Language*
DBA	*Database administrator*
DBMS	*Database management system*
DL	*Description Logic*
ER	Entidade e Relacionamento
FIPA	*Foundation for Intelligent Physical Agents*
HTML	*Hypertext Markup Language*
IA	Inteligência Artificial
JADE	*Java Agent Develop Framework*
OIL	*Ontology Inference Layer*
OWL	*Web Ontology Language*
RDF	*Resource Description Framework*
RDFS	*Resource Description Framework Schema*
RDQL	*RDF Data Query Language*
RDQL	*RDF Query Language*
SAGBD	Sistema autônomo de gerenciamento de bancos de dados
SBC	Sistemas Baseados em Conhecimento
SGBD	Sistema gerenciador de bancos de dados
SQL	*Structure Query Language*
UML	*Unified Model Language*
URI	*Uniform Resource Identifiers*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

Capítulo 1

Introdução

Este capítulo tem como objetivo apresentar uma visão geral sobre a dissertação. Será apresentada a situação que motivou o trabalho, em seguida os objetivos desejados e as contribuições que este trabalho apresenta e por fim, é descrita a estrutura dos capítulos dessa dissertação.

1.1 Motivação

As definições mais tradicionais de Sistemas Gerenciadores de Bancos de Dados (SGBD) [Date, 1999] os tratam como sistemas que permitem armazenar, modificar, excluir e recuperar dados. Entretanto, o cenário atual exigiu que esses sistemas fossem capazes de ampliar suas capacidades tanto de gerenciamento de dados quanto na automatização de tarefas que eram exaustivamente executadas através de mão de obra humana pelos Administradores de Bancos de Dados (ABD).

O uso e custo dessa mão de obra especializada além de causar substancial impacto financeiro nas organizações pode se configurar, segundo Telford [Telford, 2008], como fator crucial na geração de falhas por conta da sobrecarga atribuída às longas e repetitivas jornadas de trabalho às quais esses profissionais são expostos.

Esse contexto, em que a necessidade de automação de tarefas se alia à tendência evolutiva de softwares que são capazes de detectar, prevenir e executar tarefas, fez com que os SGBD utilizassem os recursos computacionais que a computação autônoma fornece agregando assim aos sistemas maior autonomia e capacidade de aprendizado. A essa característica dos SGBD de utilização de agentes de software com capacidade de comunicação, armazenamento e utilização de conhecimento se atribuiu a sigla: SAGBD (Sistemas Autônomos de Gerenciamento de Bancos de Dados) [Elnaffar, 2003].

Atualmente existem alguns recursos automatizados de criação de cópias de segurança (Backup) [Oracle ODM, 2008], manutenção de índices [Lohman e Lightstone, 2002], coleta de indicadores de desempenho e otimização de consultas [MSSQL Server, 2008] sendo utilizados nos SGBD comerciais mais conhecidos através da execução de tarefas por agentes de software.

Entretanto, nenhum desses SGBD possui capacidade de aprendizado e utilização de conhecimento como é visto em mecanismos criados pela iniciativa acadêmica. Essas são características do SAGBD DBSitter [Carneiro, 2004], que é capaz de armazenar decisões tomadas pelos agentes de software atuadores em um repositório de dados para uso posterior de forma reativa ou preventiva frente a uma nova situação de falha iminente.

Outros sistemas com essas características utilizam arquiteturas baseadas na comunicação entre agentes e repositórios de casos como fonte de conhecimento que são alimentados dinamicamente na medida em que novas situações são detectadas ou ações são tomadas. Porém, cada uma dessas ferramentas produz resultados, alertas e mensagens personalizados, cabendo ao administrador a tarefa de abstrair resultado semântico para correlacioná-los.

A utilização de ontologias nesse cenário pode contribuir de duas formas: através de uma ontologia de domínio que contenha os termos, mensagens, parâmetros e erros ocorridos em SGBD e outra ontologia que apresente as atribuições que cabem ao ABD.

Segundo Gruber [Gruber, 1995], uma ontologia é uma especificação formal e explícita de uma abstração, uma visão simplificada de um domínio de conhecimento. Uma ontologia é capaz de modelar um domínio de conhecimento, definindo um vocabulário comum que pode ser utilizado tanto pelos ABD quanto pelos agentes de software, a fim de estabelecer um entendimento comum sobre os conceitos e relacionamentos desse domínio.

Para isso, esse trabalho propõe a conceitualização e formalização de um modelo de ontologia do domínio e a implementação de rotinas autônomas que utilizem esse modelo. A ontologia trata o domínio de problemas em SGBD, se preocupando em apresentar os problemas e a forma como eles podem ser resolvidos através de ações reativas e proativas. Ao sistema responsável por implementar essas rotinas que acessam as ontologias deu-se o nome de OntoSitter, cujo acrônimo se baseia na junção dos termos "*Ontology*" (ontologia) e "DBSitter", sendo esse último referente ao SAGBD criado por Carneiro [Carneiro, 2004].

1.2 Objetivos

Esse trabalho, conforme exposto na seção anterior, é baseado em três conceitos da computação: Ontologias de Domínio, Computação Autônoma e Sistemas Gerenciadores de Bancos de Dados.

O objetivo desse trabalho é aprimorar os mecanismos de identificação e solução de erros ocorridos em Sistemas Autônomos de Gerenciamento de Bancos de Dados utilizando uma ontologia de domínio que retrate os problemas de administração de SGBD, ampliando assim a capacidade de auto-detecção e auto-correção de uma ferramenta acadêmica já existente para SGBD autônomos. A forma como esses conceitos foram consolidados nesse trabalho dá origem ao OntoSitter.

Para que o objetivo principal desse trabalho seja alcançado, esta dissertação se divide nos seguintes objetivos específicos:

- Realizar o levantamento do estado da arte no uso de ontologias de domínio para sistemas autônomos de gerenciamento de bancos de dados e de tarefas do ABD;
- Elaborar um conjunto de parâmetros, e mensagens de erros e soluções recorrentes em SGBD e especificá-los através da definição de uma ontologia e

- Implementar rotinas de auto-detecção e auto-correção baseadas na ontologia formalizada.

1.3 Contribuições

A principal contribuição deste trabalho será a definição de uma ontologia de domínio que contenha um conjunto de erros e soluções de problemas ocorridos em SGBD. A definição deste tipo de ontologia servirá como estrutura para a construção de bases de conhecimentos utilizadas pelos SAGBD melhor estruturadas e, conseqüentemente, melhor otimizadas.

Além do incremento no resultado comparativo da pesquisa de utilização de recursos da computação autônoma pelos SGBD e da melhoria anteriormente citada em relação às bases de conhecimento que tanto o DBSitter [Carneiro, 2004], quanto outros mecanismos autônomos de gerenciamento de bancos de dados podem usufruir, este trabalho visa comprovar que a utilização de ontologias por agentes inteligentes pode trazer inúmeras contribuições, das quais algumas são citadas no trabalho de Schumacher [Schumacher, 2003]:

1 - Compartilhamento: permitir a melhor compreensão comum sobre o domínio de conhecimento de problemas e rotinas de diversos SGBD;

2 - Reuso: utilizar definições explícitas e formais tornará mais fácil a manutenção do conhecimento e facilitará o entendimento por parte dos ABD e dos agentes de software provendo assim a reutilização da ontologia, seja ela total ou parcialmente para outras plataformas e SGBD;

3 - Estruturação da informação: conceitualizar os dados e formalizar semanticamente as informações representadas através das classes facilitam o desenvolvimento das bases de conhecimento;

4 - Interoperabilidade: ampliar o espectro de ação do DBSitter em relação ao universo de SGBD, através do uso de uma ontologia que diretamente amplia o poder de comunicação entre sistemas e

5 - Confiabilidade: representar formalmente o conhecimento diminui as possibilidades de interpretação de situações causadoras de falhas em SGBD fazendo com que as soluções sejam restringidas a um escopo melhor direcionado.

1.4 Metodologia

A forma como foi conduzido esse trabalho utilizou pesquisa bibliográfica nas áreas envolvidas, conceitualização e formalização de uma ontologia de domínio e implementação de rotinas que exemplifiquem o uso desta ontologia por agentes de software.

A metodologia Methontology proposta por López [Lópes, 1997] que apresenta cinco etapas para a criação de uma ontologia (especificação, conceitualização, formalização, integração e implementação) servirá como base para o desenvolvimento da ontologia proposta.

Como a Methontology foi criada com o propósito específico de servir como balizador para a criação da ontologia, foi necessário também utilizar uma metodologia para o desenvolvimento de rotinas que executam a comunicação entre os agentes e a ontologia. A metodologia escolhida foi a SADAAM (***S****oftware* ***A****gent* ***D****evelopment* ***A****n* ***A****gile* ***M****ethodology)* [Clynch, 2007]. O capítulo 3 apresenta a motivação para a escolha dessas metodologias.

A próxima seção apresenta em detalhes a forma como foi organizado este trabalho e de que forma serão conduzidas as tarefas.

1.5 Organização do documento

Este trabalho apresenta seis capítulos em sua totalidade, acompanhados de uma seção dedicada às referencias e outra ao anexo A. Respeitando uma seqüência lógica o trabalho possui este capítulo introdutório que apresenta objetivos, contribuições, metodologia e organização dos cinco capítulos que são descritos a seguir:

Capítulo 2: Este capítulo apresenta os conceitos básicos sobre a utilização da Computação Autônoma e Ontologias pelos sistemas computacionais e quais as utilizações desse recurso para os sistemas gerenciadores de bancos de dados. É também neste capítulo que é apresentado um comparativo entre as atuais tecnologias empregadas pelos SGBD mais conhecidos, revelando as características de autonomia desses SGBD para gerenciar e automatizar rotinas de prevenção, detecção e correção de falhas. O capítulo encerra com a apresentação da necessidade de criação de uma ontologia de domínio.

Capítulo 3: É neste capítulo que são apresentadas duas metodologias para desenvolvimento, respectivamente, de ontologias de domínio e de sistemas multi-agentes, e o capítulo encerra apresentando um padrão de comunicação entre agentes.

Capítulo 4: Enquanto os capítulos anteriores apresentam os conceitos teóricos e as etapas necessárias para construção da ontologia de domínio, fica a cargo desse capítulo, definir e formalizar essa ontologia.

Capítulo 5: Este capítulo apresenta a implementação de rotinas de acesso à ontologia de domínio por agentes autônomos. Para isso foi escolhido o SGBD Oracle e foram implementadas duas rotinas que utilizam os conceitos da computação autônoma (auto-detecção e auto-correção) para interagir com o ABD e consultar as ontologias.

Capítulo 6: Este capítulo encerra o trabalho apresentando as considerações finais sobre a utilização de ontologias de domínio e indica as perspectivas para que trabalhos futuros possam ser desenvolvidos complementando e estendendo o atual estágio de pesquisa nessa área.

Capítulo 2

Computação Autônoma e Ontologias

Este capítulo tem dois objetivos principais: apresentar um levantamento sobre a utilização dos recursos da computação autônoma e das ontologias pelos SGBD. Para isso, serão apresentados dois estudos bibliográficos sobre as atuais disponibilidades desses recursos tanto no meio acadêmico quanto no meio comercial.

2.1 Introdução

A tecnologia da informação tem avançado muito nos últimos tempos e de acordo com Hariri [HARIRI *et al*, 2006] *"uma das ambições das empresas desenvolvedoras de hardware e software é alcançar um nível onde seus artefatos possam auto-detectar seus problemas e solucioná-los de maneira a utilizar-se minimamente de esforços humanos"*.

Vislumbrando este cenário, empresas de software, como HP [Hewlett Packard, 2008], Oracle [Oracle, 2008], IBM [IBM, 2008] e Microsoft [Microsoft, 2008] que desenvolvem tecnologia na área de Sistemas Gerenciadores de Bancos de Dados (SGBD), estão buscando criar produtos voltados para uma gerência mais eficiente e para a redução do número de administradores de bancos de dados (ABD).

Com a utilização da computação autônoma, muitas tarefas que por ora são realizadas pelos ABD, [Pour, 2006] não só deixam de depender do fator humano como também passam a ser realizadas por sistemas multi-agentes (SMA).

Enquanto a indústria tecnológica desenvolve tais melhorias em seus sistemas visando a otimização de processos, as organizações e empresas que fazem uso da tecnologia empregada em SGBD para compor seus sistemas produtivos esperam que a utilização de recursos tecnológicos traga: maior valor agregado aos produtos, maior segurança e precisão na empresa, produtos e serviços de melhor qualidade, maior eficiência e produtividade, menores custos e maior controle sobre a empresa [Stair, 1998].

Diante da necessidade latente das empresas em melhorar o desempenho e agregar valor à cadeia produtiva, torna-se indispensável a utilização do fator tecnológico como aliado na busca por esses objetivos. Atualmente os SGBD que incorporam funções automáticas proporcionadas pela utilização de agentes de software podem ser em parte uma resposta para o preenchimento desta lacuna. Por outro lado, a utilização de ontologias em sistemas multiagentes já é uma realidade e pode apresentar vantagens na construção das bases de conhecimento utilizadas por esses sistemas. Portanto, o objetivo desse capítulo é apresentar o atual estágio em que encontram-se os SGBD frente a utilização dos recursos que tanto a computação autônoma quanto as ontologias podem oferecer.

2.2 Computação autônoma

Através da frase: *"Imagine um mundo onde os computadores se consertam sozinhos, antes mesmo de você saber que há um problema."* é que a empresa IBM (*International Business Machines*) [IBM, 2008] dá as boas vindas aos visitantes do endereço eletrônico que

trata sobre o assunto computação autônoma. Segundo Horn [Horn, 2007], que é um dos executivos chefes da linha de pesquisa nessa área, a automação sempre foi um dos principais fundamentos para o progresso na evolução da sociedade humana.

Fora do contexto tecnológico, o conceito que mais se aproxima da explicação de computação autônoma é o de que ela se baseia no sistema nervoso central dos seres humanos. Segundo a literatura médica [Aksoyek et al, 1999], parte do sistema nervoso humano funciona independente do cérebro e a essa peculiaridade do sistema nervoso, atribuiu-se o nome sistema nervoso autônomo (*Autonomic nervous system*).

O sistema nervoso autônomo regula as funções-chave do corpo incluindo a atividade do músculo do coração, dos músculos lisos (por exemplo, os músculos do intervalo intestinal), e de algumas glândulas. As características que determinam se um sistema é autônomo ou não também são vistas por perspectivas diferentes entre alguns autores, dos quais o Quadro 2-1 apresenta uma lista contendo essas diferentes características para computação autônoma como se segue.

Quadro 2-1: Lista de características para computação autônoma

Autor	Características para computação autônoma
Horn [Horn, 2007]	auto-conhecimento, auto-reconfiguração, auto–otimização, auto-recuperação, auto-proteção, ciência do contexto, antecipação
Kephart [Kephart, 2003]	auto-gerenciamento, auto-configuração, auto-otimização, auto-recuperação, auto-proteção
Sterritt [Sterrit, 2003]	Independência, auto-ciência, auto-gerenciamento, auto-recuperação, auto-proteção, auto-otimização
Ganek [Ganek, 2003]	auto-gerenciamento, auto-configuração, auto-otimização, auto-recuperação, auto-proteção
Agarwal [Agarwal et al, 2003]	código aberto, auto-definição, ciência do contexto, antecipação, auto-adaptação, auto-composição, auto-otimização, auto-configuração
Trumler [Trumler et al, 2003]	Ciência do contexto, antecipação, auto-recuperação, auto-configuração
Waldrop [Waldrop, 2003]	Auto-configuração, auto-otimização, auto-recuperação, auto-proteção, auto-gerenciamento.

Parashar [Parashar, 2004] defende através de um manifesto de computação autônoma que a indústria de tecnologia da informação utilize a computação autônoma partindo do

princípio que ela se baseia no funcionamento do sistema nervoso autônomo humano, que controla as funções básicas do organismo, permitindo que o cérebro possa se concentrar em funções de alto-nível.

Entretanto, na medida em que os sistemas computacionais evoluem, eles se tornam mais complexos e mais difíceis de gerenciar. Segundo Jennings [Jennings, 2001] "a administração desses sistemas é, geralmente, trabalhosa e sujeita a falhas e, com isso, o gasto das empresas com pessoal especializado é muito alto e estes se detêm em tarefas repetitivas quando poderiam estar concentrados em problemas que envolvam níveis superiores de produção". Esses fatores aliados à necessidade de se desenvolver aplicações auto-gerenciáveis, capazes de antecipar e resolver problemas têm feito com que a indústria de software se posicione de forma mais efetiva diante das perspectivas de utilização desta tecnologia.

Na indústria de software destaca-se a empresa norte americana IBM [Arellano, 2008],[Ganek, 2003],[Horn, 2007],[IBM, 2006],[Menon et al, 2003],[Océano, 2007],[Telford, 2008] onde um dos destaques é o método criado por Ganek [Ganek, 2003]. O método criado por Ganek [Ganek, 2003] determina o nível de automação dos sistemas de acordo com suas capacidades, indo assim desde o nível mais básico onde o sistema é considerado manual até o nível mais superior, onde o sistema é efetivamente reconhecido como um sistema autônomo como pode ser observado no Quadro 2-2.

Quadro 2-2: Níveis de automação dos sistemas computacionais.

Níveis	**Manual Nível 1**	**Gerenciado Nível 2**	**Preditivo Nível 3**	**Adaptativo Nível 4**	**Autônomo Nível 5**
Recursos necessários	Múltiplos recursos de sistemas geradores de dados	Consolida ferramentas de gerenciamento de dados	O sistema monitora correlações e recomenda ações	O sistema monitora, correlaciona e toma ações	Componentes integrados gerenciam dinamicamente baseados em regras e políticas de negocio
Características apresentadas	- Requer grandes capacidades e conhecimento de rotinas do operador	- Existe um responsável pela analise e tomada de ações - Aumenta a produtividade	- O responsável decide se toma ou não ações recomendadas	- Agiliza os processos e reduz ao mínimo a interação humana	- Regras de negocio e políticas são inseridas e formam uma base de conhecimento.

Com base nos níveis apresentados no Quadro 2-2 pode-se classificar se um sistema é manual, gerenciado, preditivo, adaptativo ou efetivamente um sistema autônomo. Porém, diferente de outros autores, Jeffrey Kephard [Kephard, 2003] aponta apenas quatro aspectos como fundamentais para que um sistema possa ser considerado autônomo, sendo eles:

- **Auto-configuração:** de acordo com essa definição, em um sistema autônomo que se auto-configura existem agentes de software inteligentes que configuram o sistema de acordo com regras de negócio indicadas;

- **Auto-otimização:** o aspecto de auto-otimização por sua vez é estabelecido dentro de um ambiente onde agentes de software monitoram continuamente o sistema, procurando maneiras de aumentar o desempenho das atividades do sistema;

- **Auto-recuperação:** um sistema que automaticamente detecta, diagnostica e repara problemas localizados em programas e máquinas, pode ser considerado um sistema que utiliza a característica de auto-recuperação;

- **Auto-proteção:** Contempla a capacidade de localizar pontos fracos no sistema e antecipar testes de ataque a essas vulnerabilidades além de reagir a ataques de fatores externos ao sistema.

A utilização dos conceitos de computação autônoma pelos SGBD deu origem a um novo conceito que foi apresentado no capitulo 1: Sistemas Autônomos de Gerenciamento de Banco de Dados (SAGBD), estes conceitos são apresentados na próxima seção.

2.3 Computação autônoma em SGBD

Os SGBD podem ser definidos como uma camada entre os dados e as aplicações, responsável pela execução de tarefas de manipulação, gerenciamento, concorrência e recuperação de falhas, entre outras.

O processo de alimentação desses repositórios de dados se inicia nos níveis operacionais onde os sistemas de informações transacionais obedecem aos processos definidos pelas organizações para desempenhar suas funções, passando pelos níveis gerenciais e alcançando a transformação dos dados em informações que serão utilizadas pela alta diretoria para a tomada de decisões. Essas camadas são apresentadas na Figura 2-1 onde são

indicados por setas os relacionamentos entre as aplicações e o SGBD, que nesse caso demonstra o compartilhamento da tabela de produtos por tais aplicações.

Figura 2-1: Camada entre o banco de dados e as aplicações.

Por possuir essa capacidade de armazenar desde os dados obtidos pelos sistemas transacionais até as informações sintetizadas pelos sistemas de tomada de decisão, os SGBD estão armazenando volumes cada vez maiores de dados [Heuser, 2001].

Nesse contexto, tanto a busca pela informação que apresenta um nível cada vez maior de complexidade quanto as rotinas de manutenção deste ambiente requerem um esforço maior por parte dos administradores desse recurso.

A aplicação dos conceitos de auto-configuração, auto-otimização, auto-recuperação e auto-proteção, utilizados pela computação autônoma podem ser estendidos à aplicação em sistemas gerenciadores de bancos de dados permitindo auxiliá-los na automatização de tarefas como: ajuste das áreas de memória utilizadas pelo banco de dados, criação e desfragmentação de tabelas e índices, gerenciamento de concorrência, otimização de consultas SQL, detecção e correção de falhas entre outras tarefas que até então dependiam de interação humana intensa por parte dos ABD.

2.3.1. Gerenciamento e automação de falhas em sistemas

As organizações anseiam cada vez mais por profissionais que possuam o perfil de um administrador de bancos de dados (ABD) e estejam disponíveis para jornadas de trabalho muitas vezes extensas.

No Brasil, a demanda por esses profissionais tem crescido a cada ano. Aguiar [Aguiar, 2008] afirma em seu trabalho que o perfil desses administradores precisa incluir entre suas habilidades: conhecimentos em sistemas operacionais, conhecimentos em redes, compreensão

clara de arquitetura de projetos de bancos de dados, domínio dos processos de extração, transformação e carga de dados e principalmente a proficiência em alta disponibilidade.

Manter um profissional com esse nível de especialização e por longos intervalos de tempo torna sua mão de obra cara e ao mesmo tempo expõe a empresa a um cenário iminente a ocorrência de erros que surgem pelo excesso de carga de trabalho atribuído ao ABD. A Figura 2-2 apresenta um comparativo entre as principais causas de falhas em sistemas e indica o gerenciamento como a principal causa.

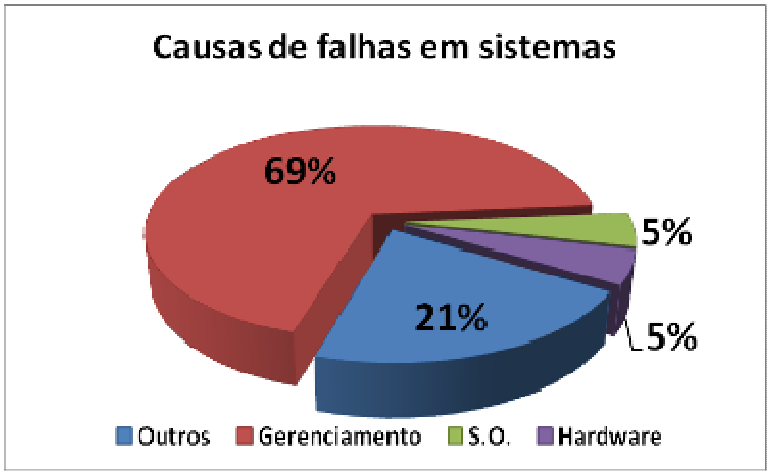

Figura 2-2: Principais causas de falhas em sistemas [Telford, 2008]

Para substituir essa mão de obra, alguns SGBD já possuem mecanismos de manutenção automatizada tanto da estrutura quanto dos dados armazenados. A utilização destes mecanismos segundo Aguiar [Aguiar, 2008] podem ser revertidas em vantagens como:

(a) Redução da carga de trabalho do ABD: automatizar as tarefas de manutenção requer supervisão mínima por parte do ABD fazendo assim com que ele se dedique a planejamento de outras atividades gerenciais;

(b) Agendamento de tarefas: quando tarefas são agendadas eletronicamente elas não dependem da disponibilidade presencial humana e podem ser realizadas a qualquer tempo e

(c) Minimização de erro humano: quando é criado um plano de manutenção automático, é minimizada a probabilidade de ocorrer erros como execução de *scripts* indevidos, selecionar bases de dados trocadas ou outros erros comuns a situação de falta de atenção na execução das rotinas.

Entretanto, todas as vantagens oferecidas por esses mecanismos os tornam reconhecidos, segundo Ganek [Ganek, 2003], como adaptativos, pois monitoram e executam

rotinas automaticamente. Porém não compõem uma base de conhecimento e não possuem capacidade de aprendizado, o que os tornaria efetivamente sistemas autônomos. As formas de como implementar arquiteturas de integração entre agentes inteligentes de software ao SGBD para dar origem ao SAGBD serão apresentadas na próxima seção.

2.3.2. Arquiteturas de implementação de SMA em SGBD

Segundo Macedo [Macedo, 2000], a implementação de agentes inteligentes de software em um SGBD deve obedecer a uma arquitetura que permita interação entre a comunidade de agentes inteligentes, também chamados de sistemas multi-agentes e o SGBD. Esta arquitetura pode ser classificada da seguinte forma:

- **Camadas (*layered*):** nessa arquitetura são implementados agentes inteligentes de software externos ao SGBD onde o programa responsável pelas funções autônomas é implementado sem modificar o SGBD;
- **Integrada (*integrated*):** as funções dos agentes inteligentes são integradas às rotinas e controles já existentes no SGBD;
- **Embutida (*built-in*):** é o modelo de arquitetura mais "invasivo", pois altera algumas funcionalidades de componentes do SGBD para que estes possam interagir com os agentes, sendo necessário o conhecimento da linguagem de desenvolvimento do SGBD e permissão para que possa ser alterada sua estrutura, o que só acontece com sistemas de código fonte livre.

A Figura 2-3 ilustra a comparação entre os modelos de arquitetura existentes na qual Macedo [Macedo, 2000] se baseou para propor a utilização da arquitetura em camadas.

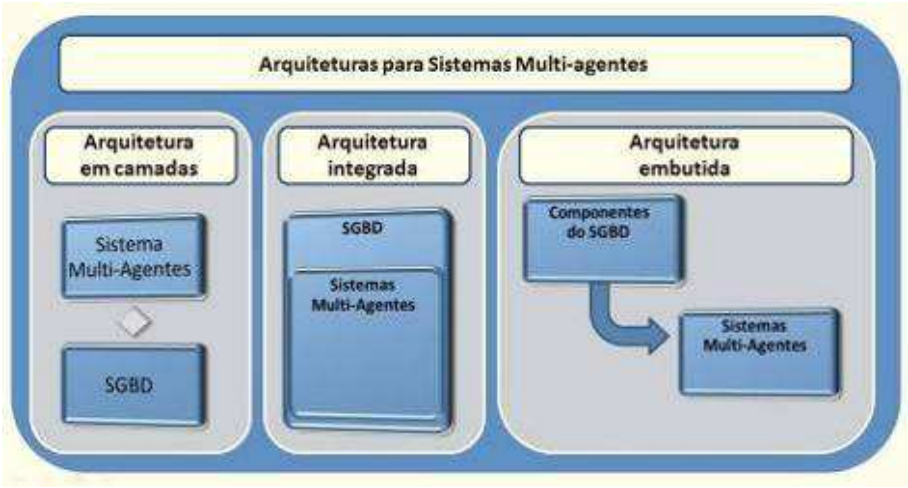

Figura 2-3: Arquiteturas de implementação de interação entre agentes e SGBD [adaptada de Macedo, 2000].

Tanto no modelo de arquitetura integrada ("*integrated*") quando no modelo embutido ("*built-in*"), os agentes precisam receber as chamadas do SGBD, interpretá-las e alterá-las e para isso esses agentes ficam embutidos no sistema, trabalhando com suas rotinas.

Por isso o modelo em camadas apresenta a vantagem de não alterar a estrutura do SGBD e sim complementá-la.

2.3.3. Trabalhos correlatos de implementação de sistemas autônomos

Esta subseção divide os trabalhos correlatos de implementação de sistemas autônomos voltados a projetos em SGBD em duas partes: iniciativas acadêmicas e iniciativas privadas.

Inicialmente serão apresentadas as iniciativas acadêmicas e um breve histórico das linhas de pesquisa que envolvem a criação de mecanismos, *frameworks* de desenvolvimento de sistemas e principalmente recursos autônomos utilizados para otimização dos SGBD, sendo esta última a que apresenta mais resultados na bibliografia sobre o assunto.

Iniciativas acadêmicas

No cenário acadêmico em relação a sistemas autônomos de gerenciamento de bancos de dados existem iniciativas que contemplam as mais variadas áreas de desenvolvimento de tecnologia para SGBD através do uso de agentes de software.

A seguir, os Quadros 2-3, 2-4, 2-5 e 2-6 apresentam trabalhos cuja relevância está ligada às capacidades dos SGBD em atender aos quesitos de auto-configuração, auto-otimização, auto-recuperação e auto-proteção.

As publicações que tratam sobre Auto-Configuração foram selecionadas atendendo a um critério preferencial para aquelas que demonstrassem mecanismos ou técnicas capazes de fazer com que os sistemas estudados pudessem ser capazes de apresentar comportamento onde regras de negócio ou alterações no ambiente fossem compreendias e levadas em consideração por esses SGBD para se adaptar ao novo cenário.

Quadro 2-3: Trabalhos recentes relacionados à auto-configuração em SGBD

Auto-Configuração				
Autor	**Publicação**	**Proposta**	**Ferramentas**	**Implementações**
[Aboulnaga, 2004]	Automated statistics collection in DB2 UDB	Coleta automatizada de estatísticas, reconfiguração do SGBD.	IBM-DB2	Duas rotinas que coletam estatísticas e uma rotina de reconfiguração de parâmetros.
[Consens, 2005]	Goals and benchmarks for autonomic configuration recommenders	Avaliar agentes embutidos em SGBD.	MSSQL 7.0 e Oracle 8	Não houve implementações, apenas testes de benchmarking
[Parek et al, 2006]	Retrofitting Autonomic Capabilities onto Legacy Systems.	Propor estudos de caso para mecanismos autônomos em camadas..	MySQL	Foram implementados quatro mecanismos autônomos em camadas.
[Mateen, 2008]	Autonomic Computing in SQL Server	Avaliar as capacidades de auto-configuração do MSSQL Server.	MSSQL 7.0	Implementadas rotinas de manutenção do SGBD.

O Quadro 2-4 apresenta a Auto-otimização nos trabalhos acadêmicos que estão intimamente ligados à capacidade dos SGBD em identificar rotinas ou recursos que podem fazer uso otimizado dos recursos computacionais.

Grande parte dos trabalhos estudados como [Macedo, 2000], lida com a otimização como uma forma de melhorar a performance das consultas efetuadas aos bancos de dados e, para isso, analisa em tempo real as consultas efetuadas e sugere melhorias ou então modifica automaticamente essas consultas.

Quadro 2-4: Trabalhos recentes relacionados à auto-otimização em SGBD

Auto-Otimização				
Autor	**Publicação**	**Proposta**	**Ferramentas**	**Implementações**
[Weikun, 2002]	Selftuning Database Technology and Information Services	Criar rotina autônoma de melhoria de índices.	MySQL	Uma rotina de melhoria de auto-indexação
[Martin, 2004]	Using Reflection to Introduce Self-Tuning Technology into DBMSs	Utilizar cálculos estatísticos para correção de parâmetros.	IBM-DB2	Rotina matemática de ajuste de parâmetros.
[Abadi et al, 2006]	Integrating Compression and Execution in Column-oriented Database Systems	Representar matematicamente as vantagens de utilizar auto-compressão de dados.		Não houve implementação, foram demonstradas matematicamente as vantagens.
[Martin et al, 2007]	The WSDM of Autonomic Computing: Experiences in Implementing Autonomic Web Services	Apresentar um estudo bibliográfico sobre os sistemas autônomos da IBM.	IBM-OASIS	Implementação de rotinas autônomas que deram origem ao projeto OASIS.

Afirmar que um sistema é capaz de manter mecanismos de Auto-recuperação consiste em compreender que ele é capaz de detectar, diagnosticar e reparar falhas que o impossibilitem de se manter em funcionamento.

Para um SGBD as três características são fundamentais, porém, quanto maior a capacidade deste sistema de realizar o auto-diagnóstico de falhas iminentes o torna mais resistente a falhas futuras e, conseqüentemente, mais independente de ações humanas para corrigir ou alterar o ambiente no qual ele está inserido.

O Quadro 2-5 apresenta exemplos acadêmicos de estudos onde a Auto-Detecção trabalha em conjunto com a habilidade de Auto-Recuperação.

Quadro 2-5: Trabalhos recentes relacionados à auto-recuperação em SGBD

Auto-Recuperação				
Autor	**Publicação**	**Proposta**	**Ferramentas**	**Implementações**
[Liu et al, 2004]	The design and implementation of a self-healing database system	Apresentar os riscos de rotinas maliciosas de auto-recuperação	MSSQL 7.0 e MySQL	Por se tratar de um estudo bibliográfico, o trabalho não apresenta implementações.
[Wang, 2007]	Evaluating the survivability of Intrusion Tolerant Database systems and the impact of intrusion detection deficiencies	Avaliar o desempenho de SGBD que utilize mecanismos de detecção de intrusão.	PostGreSQL e MySQL	Realizados testes de desempenho com rotinas em PL/SQL
[Ghosh et al, 2007]	Self-healing systems - survey and synthesis.	Revisar a bibliografia.	Por se tratar de um estudo bibliográfico, o trabalho não apresenta implementações.	Não foram desenvolvidas rotinas, apenas revisão bibliográfica.
[Montani, 2008]	Achieving self-healing in service delivery software systems by means of case-based reasoning	Utilizar um sistema baseado em arquitetura de raciocínio baseado em casos (RBC).	PostGreSQL	Desenvolvidas duas rotinas de busca e atualização do RBC.

O Quadro 2-6 revela que um SGBD com capacidade de Auto-proteção é capaz de manter tanto características reativas quanto proativas e nesse contexto, utilizar uma base de conhecimentos amplia o histórico de casos com semelhança de acontecimento e assim torna mais fácil a classificação dos eventos causadores de falhas e suas ações necessárias para correção.

Quadro 2-6: Trabalhos recentes relacionados à auto-proteção em SGBD

Auto-Proteção				
Autor	**Publicação**	**Proposta**	**Ferramentas**	**Implementações**
[Gray, 2004]	The next database revolution	Revisar a bibliografia sobre avanços em auto-proteção.	O estudo não utilizou ferramentas específicas.	Não houve implementações.
[Thereska et al, 2006]	Informed data distribution selection in a self-predicting storage system	Apresentar a capacidade de um SGBD em se antecipar a problemas	FireBird	Criadas rotinas em PG/SQL para o SGBD FireBird.
[Holze, 2007]	Towards workload shift detection and prediction for autonomic databases.	Revisar bibliografia sobre desempenho de SGBD expostos a riscos.	O estudo não utilizou ferramentas específicas.	Não houve implementações.
[Goel et al, 2007]	Automatic high-performance reconstruction and recovery.	Avaliar os riscos da detecção de falsos riscos.	MySQL	Desenvolvida rotina em Java para o SGBD MySQL.

Uma proposta acadêmica de destaque em relação a sistemas autônomos de gerenciamento de bancos de dados é o sistema DBSitter [Carneiro, 2004] que utiliza o modelo em camadas para interagir com o SGBD e o ABD de forma não invasiva às rotinas do SGBD ao mesmo tempo em que mantém um esquema interno de comunicação entre agentes de software. Essa arquitetura possui uma particularidade que a torna diferente dos sistemas comerciais: a capacidade de aprendizado e o modo como foi proposto ainda permite que ele seja integrado a esses sistemas sem interferir nas rotinas internas. O Capítulo 5 apresenta a forma como nossa proposta, o OntoSitter, pode ser utilizado junto ao DBSitter.

Além dessa proposta, iniciativas apresentam uma convergência nesse sentido, propondo desde a criação de frameworks de desenvolvimento de SAGBD [Maciel, 2006], passando por propostas de arquiteturas [Pour, 2006] e implementação de mecanismos de

verificação de desgaste estrutural em SAGBD [Yoshihiro, 2002], mas a área de maior destaque é a de *tunning* de índices de tabelas através do uso de agentes de software [Macedo, 2000].

Iniciativas privadas

Nesta subseção serão consideradas as iniciativas das empresas privadas para a obtenção de mecanismos autônomos de gerenciamento de bancos de dados e para isso serão considerados os quatro aspectos citados por Kephart [Kephart, 2003] como princípios da computação autônoma para comparar as atuais tecnologias disponíveis para SGBD em relação a esses princípios. O Quadro 2-7 apresenta um paralelo entre esses princípios e um exemplo de situação em que pudesse ser utilizada a tecnologia da computação autônoma no auxílio a cada situação proposta para avaliar quais sistemas têm condições de atender aos princípios da computação autônoma.

Quadro 2-7: Características e aspectos da computação autônoma

Característica	Descrição	Comportamento desejado
Auto-configuração	O SGBD possui mecanismos autônomos de ajuste da configuração de seus parâmetros de acordo com as mudanças do ambiente em que está inserido.	**(1)** Ao diminuir o espaço em disco o sistema reconfigura o parâmetro de tamanho de banco de dados ou emite aviso ao ABD?
Auto-otimização	O SGBD monitora e ajusta automaticamente recursos do sistema e da máquina.	**(2)** São examinadas as consultas SQL, a complexidade e custo de processamento?
Auto-recuperação	O SGBD é capaz de manter-se disponível e recuperar o status anterior de seus objetos após falhas.	**(3)** Se houver queda de energia e corromper o arquivo de *log* o SGBD reinicia e recupera-se ?
Auto-proteção	O SGBD é capaz de se antecipar, detectar e se proteger de falhas iminentes.	**(4)** O SGBD monitora o ambiente onde está inserido?

Frente a essas situações foi compilado através do Quadro 2-8 o resultado da comparação entre as capacidades que esses SGBD possuem para atender aos requisitos da computação autônoma.

Quadro 2-8: Computação autônoma aplicada a SGBD

	Aspectos fundamentais			
Software	**(1)**	**(2)**	**(3)**	**(4)**
Oracle 11g [Oracle ODM, 2008]	◒	◒	●	◒
IBM DB2 9.5 [IBM DB2, 2008]	◒	◒	◒	◒
Microsoft SQL 2008 [MSSQL Server, 2008]	◒	◒	●	◒
Progress 10.1B [Progress, 2008]	○	◒	◒	◒
Sybase SQL 10 [Sybase, 2008]	◒	◒	○	◒
PostgreSQL 8.3.1 [PostgreSQL, 2008]	◒	◒	○	○
FireBird 2.0.3 [Firebird, 2008]	○	○	○	○

Aspectos:

1 - Auto-Configuração
2 - Auto-Otimização
3 - Auto-Recuperação
4 - Auto-Proteção

Legenda:

● Atende

◒ Atende Parcialmente

○ N.A. - Não atende

Para a criação desta figura de comparações foi levado em consideração o critério de avaliação dos SGBD onde se atribuiu nota máxima ao aspecto que é atendido sem que exista a necessidade de interação humana.

A nota “atende parcialmente” foi dada para os aspectos que são atendidos através de agentes de software, mas que sofrem algum tipo de interação humana para a tomada de decisões e, finalmente, a nota “N.A. - Não atende” foi dada para os aspectos que não são atendidos nem por rotinas autônomas nem por agentes de software (esse caso pode indicar tarefas que dependam exclusivamente de interação humana ou então são situações onde o SGBD fica incapacitado de operar).

A partir dos dados coletados com essa comparação, este trabalho mantém o foco sobre os detalhes de funcionamento das rotinas autônomas nos SGBD Oracle [Oracle ODM, 2008], DB2 [IBM DB2, 2008] e MSSQL [MSSQL Server, 2008]. Os três SGBD aqui comparados possuem mecanismos de auto-otimização e auto-proteção que funcionam de maneira semelhante, ou seja, baseados em repositórios alimentados pela coleta de dados sobre a performance do sistema para posterior tomada de decisão ou geração de relatórios.

SGBD Oracle

O SGBD Oracle a partir da versão 9 [Oracle ODM, 2008] utiliza o mecanismo chamado AWR (*Automatic workload repository*) que é responsável por armazenar informações sobre performance do SGBD realizando monitoramentos em intervalos de tempo pré-estimados e armazenando esses resultados em um repositório. O Quadro 2-9 apresenta um

exemplo do comando em SQL que é responsável pela coleta desses dados. Este comando executa em primeira instancia a alteração dos parâmetros, para que o SGBD passe a atualizar os dados sobre performance a cada 30 segundos. Em seguida cria um identificador para cada uma das atualizações, de maneira a se tornar possível a localização dessas atualizações futuramente.

Quadro 2-9: Comando SQL que realiza a coleta de informações

```
BEGIN
   DBMS_WORKLOAD_REPOSITORY.modify_snapshot_settings
   (retention=>43200,interval => 30);
END;

EXEC DBMS_WORKLOAD_REPOSITORY.create_snapshot;

BEGIN
   DBMS_WORKLOAD_REPOSITORY.drop_snapshot_range ( low_snap_id => 22,
   high_snap_id => 32);

END;
```

Feita a coleta desses indicadores de performance, o SGBD utiliza outro mecanismo chamado ADDM (*Automatic database diagnostic monitor*) que executa três processos: análise do conteúdo do AWR, aconselhamento para os itens considerados fora do padrão e armazenamento do aconselhamento para cada situação. O Quadro 2-10 apresenta um exemplo de recomendação de alteração de parâmetros do SGBD de modo que o aumento da capacidade de armazenamento em disco da memória secundária do SGBD poderá ter um incremento de velocidade na leitura e escrita. O comando apresentado nesse Quadro é dividido em quatro partes: a) Detecção do problema através do comando "*FINDING*"; b) Recomendação de correção de parâmetros pelo comando "*RECOMMENDATION*"; c) Qual ação deve ser tomada para correção do problema pelo comando "ACTION" e d) Quais os sintomas que causaram o problema através do comando "SYMPTOMS THAT LED TO THE FINDING".

Quadro 2-10: Recomendação de incremento de memória

```
FINDING 1: 59% impact (944 seconds)
The buffer cache was undersized causing significant additional read
I/O.

RECOMMENDATION 1: DB Configuration, 59% benefit (944seconds)

ACTION: Increase SGA target size by increasing the value of
parameter "sga_target" by 28 M.

SYMPTOMS THAT LED TO THE FINDING: Wait class "User I/O" was
consuming significant database time. (83% impact [1336 seconds])
```

Para que essa recomendação seja atendida serão necessárias ações que modifiquem o hardware, a configuração do SGBD ou até mesmo os esquemas conceituais do BD e para isso são necessárias rotinas que podem ser agendadas previamente através de planos de manutenção ou então que sejam executadas por agentes de software.

A utilização de um ambiente em que agentes de software interagem desde a coleta passando pela escolha do plano de manutenção e chegando à execução do plano é o cenário ideal, porém ainda não implementado pelas soluções comerciais como é o caso da iniciativa acadêmica DBSitter apresentado na seção anterior.

O SGBD Oracle [Oracle ODM, 2008] utiliza também o agendamento de rotinas de manutenção para que o STA (*SQL Tunning Advisor*) possa executar ajustes indicados pelo ADDM. O Quadro 2-11 apresenta a execução de uma instrução SQL que fornece um relatório indicador de ajustes necessários a serem realizados no SGBD. A exemplo do quadro anterior, este quadro utiliza os mesmo quatro itens para localizar os problemas e recomendar uma ação que corrija essa situação.

Quadro 2-11: Execução de comando SQL para gerar relatório de estatísticas

```
select dbms_sqltune.report_tuning_task('emp_dept_tuning_task') as
recommendations from dual;

---------------------

1- Statistics Finding-----------
Table "SCOTT"."EMP" and its indices were not analyzed.

Recommendation--------------
Consider collecting optimizer statistics for this table and its
indices.

execute dbms_stats.gather_table_stats(ownname => 'SCOTT', tabname =>
'EMP', estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE, method_opt
=> 'FOR ALL COLUMNS SIZE AUTO', cascade => TRUE)
```

SGBD IBM DB2

O SGBD IBM DB2 [IBM DB2, 2008] utiliza conceitualmente a mesma solução adotada pelo SGBD Oracle através de mecanismos de "*aconselhamento*" que sugerem os melhores parâmetros de configuração do SGBD de acordo com as informações indicadas pelo ABD no momento de instalação do sistema.

Um desses mecanismos é chamado de DB2 *Configuration Advisor* e apresenta

inicialmente valores de parâmetros padronizados de instalação do SGBD, os valores atuais desses parâmetros e as sugestões de parâmetros indicados pelo mecanismo do DB2 são visualizados pelo ABD. Após coletar as informações fornecidas pelo ABD no momento de instalação do SGBD o mecanismo atualiza os parâmetros ajustando-o para obter um desempenho de acordo com as especificações fornecidas.

O IBMDB2 apresenta ainda indicadores do nível de "saúde" do SGBD através de monitores que coletam informações sobre os objetos como bases de dados e espaços em tabelas, qualificando os diferentes níveis de objetos que estão cobertos por esse mecanismo de monitoramento. (e.g.: *Databases, Table Spaces, User Spaces*.)

Outro mecanismo pertencente a essa estrutura é o *Health Center* que combina o monitoramento automático do SGBD com as informações prestadas pelo ABD sobre intervalos de monitoramento e quais objetos devem ser monitorados, uma vez que esse serviço também pode resultar em significativa perda de desempenho do SGBD. A Figura 2-4 apresenta a tela deste mecanismo onde podem ser observados os percentuais de utilização de alguns recursos do SGBD.

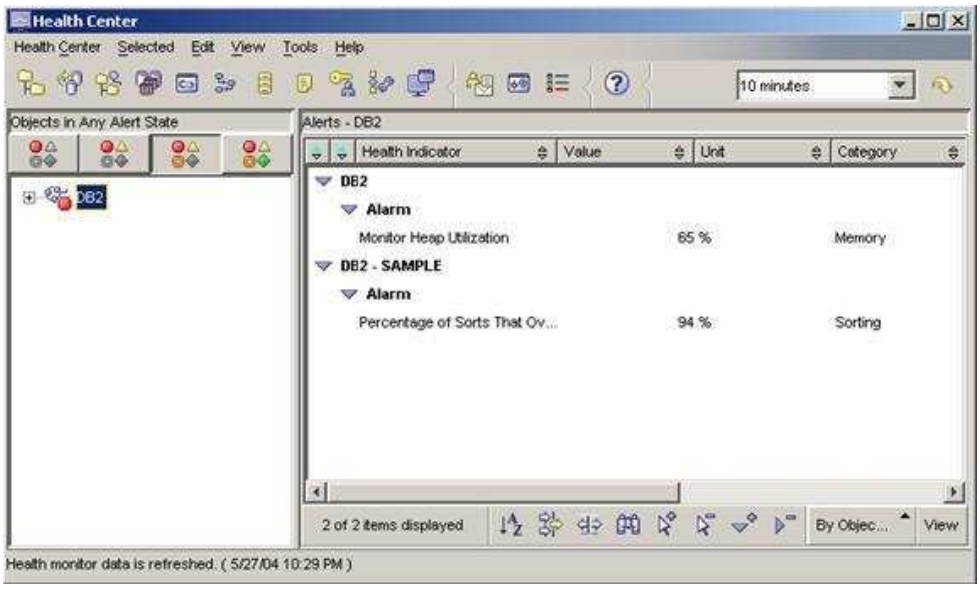

Figura 2-4: Mecanismo de monitoramento de objetos do IBMDB2

A combinação de mecanismos de aconselhamento com os mecanismos de monitoramento do SGBD tornam muitas atividades do ABD dispensáveis por conta da automatização dessas tarefas, porém, ainda assim, a interação ABD e SGBD ainda é bastante necessária nesse sistema e fica evidente especialmente nas etapas de instalação e configuração do SGBD, necessitando assim de um profissional especialista em parametrização e otimização.

SGBD MSSQL

O SGBD Microsoft SQL Server [MSSQL Server, 2008] utiliza o DTA (*Database engine tunning advisor*) para coletar os dados de performance e realizar ajustes no sistema. Esse mecanismo foi criado ainda no projeto AutoAdmin [AutoAdmin, 2008] criado para o sistema na versão MSSQL2000 e mantém as mesmas propriedades: fornecer ao ABD um assistente de configurações e um mecanismo cujo objetivo é otimizar o custo de processamento das consultas em SQL.

Diferente dos sistemas citados anteriormente, o MSSQL ainda não possui um mecanismo de auxilio à otimização de instalação que permita ou que indique uma parametrização capaz de utilizar de forma mais eficiente os recursos computacionais disponibilizados pelo servidor. O sistema apresenta em suas telas iniciais de instalação uma análise de 7 regras básicas de compatibilidade de recursos para sua instalação.

Os recursos autônomos do MSSQL ainda não são perceptíveis na versão atual e comparativamente aos outros sistemas aqui mencionados nesse trabalho ele apresenta apenas vantagens em relação à interface oferecida ao ABD, o que é uma característica dos sistemas da Microsoft. Na Figura 2-5 são demonstrados os relatórios de acompanhamento de performance.

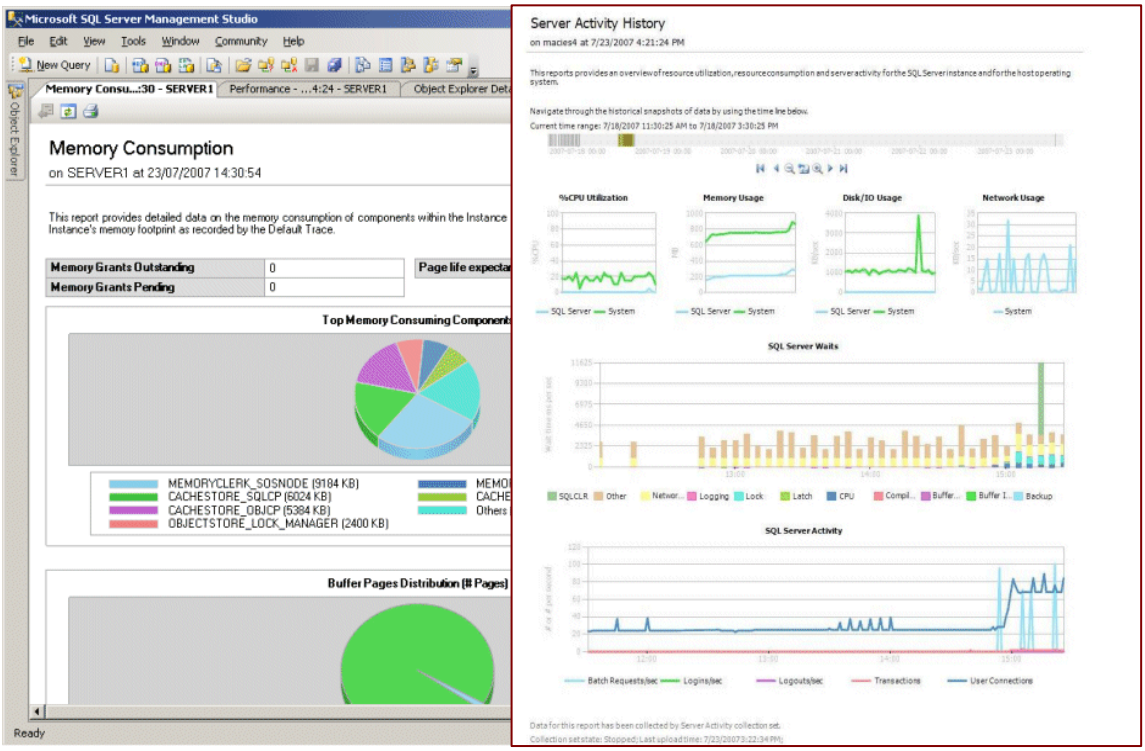

Figura 2-5: Relatórios de performance do MSSQL2008 [MSSQL Server, 2008]

Essa seção apresentou as iniciativas das empresas de SGBD para a computação autônoma e apresentou algumas características dos SGBD Oracle, DB2 e MSSQL. Embora esses SGBD possuam recursos que auxiliem a administração do SGBD por parte do ABD esses sistemas ainda estão distantes de atingirem o nível desejado que se espera de um sistema autônomo. A próxima seção apresenta o estudo bibliográfico sobre ontologias e qual o impacto exercido nos SGBD.

2.4 Ontologias

Historicamente o termo ontologia tem origem no grego "ontos", ser, e "*logos*", palavra. O termo original é a palavra aristotélica *"categoria"*, que pode ser usada para classificar alguma coisa. Aristóteles apresenta categorias que servem de base para classificar qualquer entidade e introduz ainda o termo *"differentia"* para propriedades que distinguem diferentes espécies do mesmo gênero. A conhecida técnica de herança é o processo de mesclar *differentias* definindo categorias por gênero [Marcones, 1997] .

A definição de ontologia em Ciência da Computação é dada como a especificação formal e explícita de uma conceituação compartilhada [Gruber, 1993]. A conceituação indica um modelo que identifique os conceitos relevantes de algo do qual se queira explicitar; formal se refere ao fato de que estruturar uma ontologia a torna capaz de ser interpretada pelo raciocínio lógico de uma máquina; e compartilhada revela que o conteúdo armazenado em uma ontologia é fruto de conhecimento coletivo.

Guarino complementa essa definição ao afirmar que uma ontologia é uma especificação parcial e explícita cujo objetivo é aproximar a estrutura do mundo real definida por uma conceituação [Guarino, 1998].

A busca e utilização de ontologias pela Ciência da Computação surgiram inicialmente na área de Inteligência Artificial, visando criar segundo Gava e Menezes, representações semânticas capazes de ir além da descrição de simples instâncias do domínio [Gava e Menezes, 2003]. Atualmente, desenvolver vocabulários contendo conceitos relativos à domínios de aplicação é uma tarefa que se expandiu por várias outras áreas da Ciência da Computação.

Segundo [Guarino, 1998], as ontologias são classificadas em diferentes tipos de acordo com seus conteúdos. Assim sendo, as ontologias podem se dividir em: **(i) ontologias genéricas:** descrevem conceitos bastante gerais, tais como, espaço, tempo, matéria, objeto,

evento, ação, etc., que são independentes de uma situação ou domínio particular; **(ii) ontologias de domínio:** expressam conceituações de domínios particulares, descrevendo o vocabulário relacionado a um domínio genérico, tal como Medicina e Direito; **(iii) ontologias de tarefas:** expressam conceituações sobre a resolução de problemas, independentemente do domínio em que ocorram, isto é, descrevem o vocabulário relacionado a uma atividade ou tarefa genérica, tal como, diagnose ou vendas; **(iv) ontologias de aplicação:** descrevem conceitos dependentes do domínio e da tarefa particulares. Estes conceitos freqüentemente correspondem a papéis desempenhados por entidades do domínio quando da realização de uma certa atividade; **(v) ontologias de representação:** explicam as conceituações que fundamentam os formalismos de representação de conhecimento.

Esse trabalho, portanto, utilizará daqui por diante o termo ontologia, sempre fazendo menção à classificação de ontologias de domínio, uma vez que esse conceito representa uma ontologia que possui um vocabulário de conceitos, definições e suas propriedades, e é representada por um modelo formal de relações entre esses conceitos e os axiomas capazes de apresentar de maneira clara e não ambígua o conhecimento do domínio.

2.4.1. Ontologias e SGBD

Para compreender a função de uma ontologia junto a um SGBD, é necessário além da definição do conceito, elucidar que uma ontologia é uma representação formal de um dado conceito ou domínio. A Figura 2-6 apresenta uma ilustração que classifica ao lado esquerdo da barra vertical os objetos que não são uma ontologia, enquanto os objetos à direita são as propriedades que compõem uma ontologia.

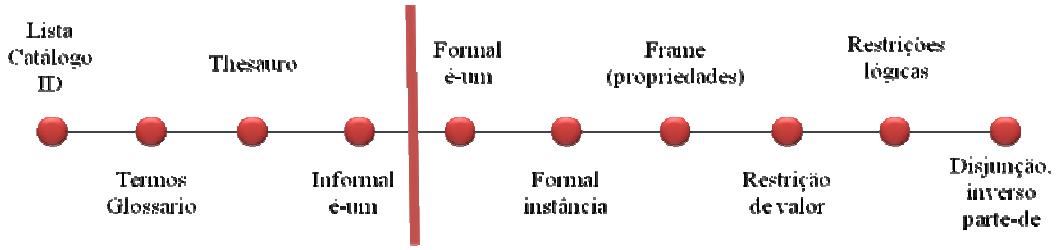

Figura 2-6: Objetos que representam ou não uma ontologia [McGuiness et al, 2000].

O que define a diferença entre listas, glossários e termos, de uma ontologia é o fato de que uma ontologia precisa atender a um grau mínimo de formalidade e isso é conseguido através da verificação de seus axiomas, que são constituídos pelas restrições que utilizam as propriedades da ontologia criada.

Atualmente existem várias formas de pesquisas de termos referentes tanto ao domínio dos SGBD quando das rotinas executadas para o gerenciamento desses sistemas. Em sua maioria, essas fontes de dados podem ser encontradas através de consultas manuais na internet à glossários [Glossarist, 2008], enciclopédias [TechWeb, 2008] e [Wikipedia, 2008].

Existe ainda a possibilidade de utilizar mecanismos de extração de informação (EI) para automatizar essa tarefa de busca e classificação da informação em documentos e na internet. Vários trabalhos apresentam o uso de agentes de software para a extração, classificação e armazenamento do conhecimento, porém, Freitas [Freitas, 2002] indica uma das dificuldades nessa classificação como sendo o fato de que "*as próprias páginas possuem, às vezes, conteúdo ambíguo e vago, e isso pode fazer parte da própria informação, seja por seu conteúdo (como, por exemplo, em poesia), ou pela localidade do vocabulário empregado, mantendo o problema da linguagem natural"* [Freitas, 2002].

Accioly apresenta em seu trabalho [Accioly et al, 2002] duas propostas de ambientes para aquisição de conhecimento automático baseadas em engenharia reversa que parte da informação armazenada em SGBD relacionais para dar origem a uma ontologia inicial.

O *Karlsruhe Ontology and Semantic Web Infrastructure* (KAON) é um *plug-in* de conexão e mapeamento de bancos de dados relacionais para uma ontologia e tem como objetivo extrair instâncias e relacionamentos. A conexão é realizada através de uma conexão *Java Database Connectivity* (JDBC). A vantagem deste *plug-in* é a capacidade automática de extração do esquema do BD. Fica a cargo do usuário a definição do mapeamento de tabelas e da integridade referencial para a ontologia [Kaon, 2001].

Outro tipo de aquisição automatizada baseada no processo de extração de ontologias a partir de um banco de dados relacional é realizado pelo mecanismo chamado *Domain Ontology Management Environment* (DOME), onde as fontes de dados são os sistemas legados dos quais são extraídas as ontologias através de engenharia reversa para extração de uma ontologia inicial [O´Brien, 2000]. A Figura 2-7 apresenta a relação entre as ontologias, os modelos entidade relacionamento e os esquemas de bancos de dados.

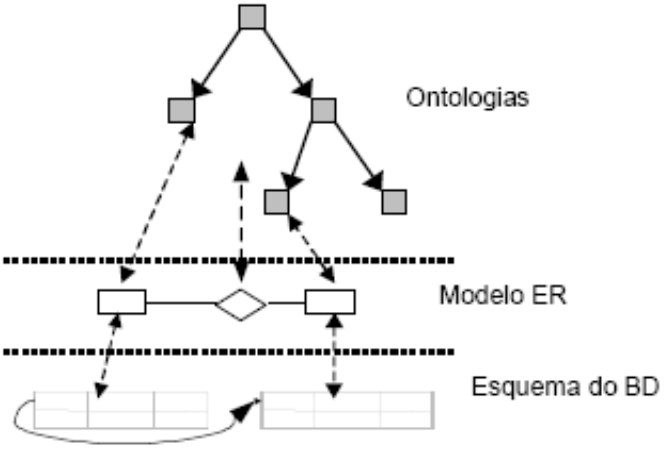

Figura 2-7: Engenharia reversa para gerar ontologias de BD relacionais [O´Brien, 2000]

Entretanto, não foi localizada durante a pesquisa bibliográfica que compõe este trabalho uma ontologia específica do domínio de SGBD ou das rotinas de trabalho do administrador de bancos de dados (ABD) que seria de extrema utilidade para direcionar as ações dos agentes de software que utilizassem os dados de uma base de conhecimentos baseada nessa ontologia.

A iniciativa que mais se aproxima dessa necessidade (embora não tenha relação nem formatação ligada a uma linguagem de ontologia) foi identificada como DMBOK (*Data management book*), e se trata de um guia publicado pela fundação DAMA (*Data Management Association*) [Dama, 2008]. Seu objetivo é criar um modelo que contenha definições padronizadas para funções do gerenciamento de dados, papéis, regras e terminologias comuns aos ABD e servirá como base para a construção da ontologia proposta nesse trabalho. Como a quantidade de informação referente a essas rotinas contida nesse documento não é extensa[1], optou-se pela extração e classificação manual da ontologia proposta.

2.5. Considerações

O objetivo principal deste capítulo foi apresentar o levantamento bibliográfico sobre a utilização dos conceitos referentes à computação autônoma e ontologias e em que estágio está a aplicabilidade destes conceitos aos SGBD.

Entretanto, a pesquisa bibliográfica revelou grande quantidade de ontologias para os mais variados domínios de aplicação, com enfoque, sobretudo para a área médica. Ontologias

[1] O conjunto completo de atribuições de tarefas de administração e gerenciamento de um SGBD pelo ABD segundo o PMBOK é apresentado no Anexo A.

de domínio que designam termos utilizados no gerenciamento de bancos de dados não foram encontradas, por isso optou-se pela criação de uma ontologia fundamentada nos conceitos da área de engenharia de software para problemas e erros. Essa ontologia será apresentada no Capítulo 4.

Uma tendência observada é a de que as empresas privadas responsáveis pelos SGBD proprietários estão incorporando aos sistemas os mecanismos autônomos antes externos às suas aplicações. Essa mudança de paradigma além de incrementar de forma significativa as rotinas de um SGBD também muda o modelo de utilização de sistemas multiagentes que outrora era realizado através da arquitetura em camadas e passa a utilizar o modelo embutido.

Se por um lado surge a vantagem de não depender de um sistema intermediário, por outro se restringe à utilização das rotinas automatizadas a uma solução proprietária que pode ou não atender às necessidades específicas de determinados ambientes que antes poderiam ser melhor gerenciados através de mecanismos independentes.

Enquanto a iniciativa privada busca isolar dentro dos próprios aplicativos as soluções para a autonomia, a iniciativa acadêmica pesquisa e cria sistemas que atendam ao modelo em camadas, permitindo assim que mecanismos não invasivos sejam criados alcançando maior independência tanto de plataforma quanto de sistema.

Sob essa ótica, pode-se afirmar que as pesquisas acadêmicas têm apresentado melhores resultados do que as aplicações em uso, uma vez que dispõem de cenários onde os SGBD podem executar planos de manutenção totalmente independentes da interação humana e adquirir, armazenar e utilizar conhecimentos para tomada de decisão frente a falhas.

A forma como esses conhecimentos são adquiridos, armazenados e utilizados é o que tornará os SAGBD mais ágeis e preemptivos. Entretanto, para isso, é necessário que esses sistemas compreendam a linguagem utilizada pelos ABD no tratamento de suas rotinas.

Capítulo 3

Metodologias e Padrões Adotados

Este capítulo fundamenta a escolha da metodologia utilizada para o desenvolvimento da ontologia de domínio de problemas e gerenciamento de SGBD e em seguida a metodologia para o desenvolvimento do sistema multiagentes que fará acesso a essa ontologia e a interação com o administrador de bancos de dados. Por fim, apresenta o padrão adotado para comunicação entre os agentes de software.

3.1 Introdução

Nos últimos anos, houve um incremento exponencial dos dados disponíveis tanto na internet quanto pelas organizações e na maneira como esses dados são acessados e isso criou naturalmente a necessidade de ampliar as técnicas de organização da informação. Essas técnicas são responsáveis por buscar melhorias no tratamento de dados e atuar na seleção, processamento e recuperação dos dados.

Para que uma ontologia seja criada por especialistas e defina as regras que regulam a combinação entre termos e relações em um domínio do conhecimento é necessário utilizar uma metodologia que otimize o processo de criação e permita a edição e manutenção desta ontologia.

Assim, na Seção 3.2 a metodologia escolhida para guiar os passos do desenvolvimento da ontologia desse trabalho é apresentada. Uma vez que para interagir com essa ontologia, este trabalho utiliza rotinas baseadas em agentes inteligentes que se comunicam e mantêm comunicação com o administrador de bancos de dados, a seção 3.3 vai tratar sobre algumas metodologias disponíveis para o desenvolvimento de sistemas multiagentes e também justifica a escolha de uma metodologia recentemente desenvolvida.

3.2 Methontology (Metodologia para ontologias)

A fim de amparar a formalização dos modelos conceituais de ontologias, algumas metodologias foram propostas. O estudo de Corcho e Gomez-Pérez [2000] apesar de antigo, apresenta uma análise destas metodologias e das ferramentas utilizadas em seu desenvolvimento. Este estudo compara não só as metodologias e as ferramentas, como também as principais linguagens relatadas na literatura. O trabalho minucioso apresenta um cenário de necessidade de integração entre o desenvolvimento, a avaliação, a evolução e a gerência da ontologia, o que em síntese representa o desejo de integrar os componentes da tecnologia que estão atualmente disponíveis.

No Quadro 3-1, são apresentadas três metodologias, com seus respectivos pontos positivos e pontos negativos que ajudaram a fundamentar a escolha da metodologia **Methontology** [Fernandez-Lopez, 1999] como a metodologia indicada para o desenvolvimento de ontologia de domínio para esse trabalho.Foram escolhidas essa metodologias pelo fato de serem as mais citadas na literatura e por possuírem características próximas às necessidades de desenvolvimento de uma ontologia de domínio.

Quadro 3-1: Quadro de comparação de metodologias

Metodologia	Descrição	Ponto Positivo	Ponto Negativo
Cyc	Codifica o conhecimento das fontes e, quando já existe conhecimento suficiente na ontologia, novo consenso é obtido por ferramentas de linguagem natural [Lenat, 1990].	Fornece suporte intenso à fase de aquisição do conhecimento, além de ser direcionada a ontologias de domínio.	A fase de gerenciamento do projeto não aparece como uma fase no ciclo de vida dessa metodologia, assim como a modelagem conceitual e a manutenção.
Methontology	Constrói-se uma ontologia por reengenharia sobre outra ontologia. A construção utiliza o conhecimento do domínio e as atividades principais são: especificação, conceitualização, formalização, implementação e manutenção [Fernandez-Lopez, 1999].	Apresenta etapas de desenvolvimento bem definidas que englobam desde a aquisição do conhecimento aos testes com axiomas.	Por ser focada para ontologias de domínio, não cobre os outros tipos, embora esse ponto negativo tenha sido revertido em um ponto positivo a favor do OntoSitter, que utiliza uma ontologia de domínio.
Maedche	Uma ontologia genérica é convertida em um modelo computável; especificam-se textos e obtêm-se conceitos do domínio a partir das fontes disponíveis; removem-se conceitos genéricos, e apenas o domínio permanece. [Kaon, 2001].	Apresenta ao final das etapas uma ontologia bem definida e de baixa manutenção por conta do processo incremental e interativo de criação.	Entre as metodologias comparadas, é a que menos trata a questão de integração e reutilização de ontologias, o que a torna inferior em relação às anteriores.

Os fatores que predominaram na escolha da Methontology [Fernandez-Lopez, 1999] para o desenvolvimento de uma ontologia de domínio de problemas em SGBD foram o fato de que essa metodologia é que suas etapas distingüem claramente as atividades e os artefatos esperados do processo de criação de uma ontologia. As etapas são descritas abaixo:

i. Atividades de Gerenciamento de Projeto: incluindo planejamento, controle e garantia de qualidade. No planejamento são definidas quais tarefas serão desenvolvidas; no controle é garantido que as tarefas planejadas serão executadas e a garantia de qualidade busca desenvolver o produto como estabelecido pelos seus requisitos.

ii. Atividades de Desenvolvimento Orientado por Etapas: inclui especificação, conceitualização, formalização e implementação. A especificação define o porquê da ontologia estar sendo construída (utilização e usuários finais). A conceitualização estrutura o domínio de conhecimento em um modelo conceitual. A formalização transforma o modelo conceitual em um modelo formal ou semi-computável. A implementação transforma modelos computáveis em linguagens computacionais.

iii. Atividades de Suporte: inclui aquisição de conhecimento, avaliação, integração, documentação e gerenciamento de configurações. Na aquisição de conhecimento é estudado o conhecimento de um determinado domínio. Na avaliação são realizadas comparações técnicas das ontologias, relacionadas com software e documentação.

A integração de ontologias busca desenvolver uma nova ontologia a partir de outras já disponíveis. A documentação é a explicação completa e exaustiva de todos os recursos e fases existentes. Gerenciamento de configurações registra todas as alterações (versões) de documentação, software e ontologias. A criação dos componentes da ontologia é realizada, principalmente, na etapa de atividades de desenvolvimento orientado, especificamente, na conceitualização e formalização do conhecimento. De posse da metodologia de formalização, se faz necessário especificar a linguagem utilizada para a criação da ontologia.

Estando a metodologia definida, serão apresentadas a seguir quatro etapas para o desenvolvimento da ontologia de domínio (especificação, conceitualização, formalização e implementação), assim, essa ontologia será capaz de contemplar a definição de compartilhamento e reuso.

3.2.1. Etapa 1 – Definição de Escopo

A primeira etapa do desenvolvimento de uma ontologia de domínio é a especificação, e para isso é necessário buscar os conhecimentos que serão utilizados por essa ontologia. A

aquisição de conhecimento (AC) para sistemas baseados em conhecimento (SBC) pode em síntese ser definida como um processo onde o conhecimento advindo de várias fontes documentadas ou não é organizado e melhor explicitado.

Segundo [Regoczei e Hirst, 1994]. AC é o processo total de apreender o conhecimento e transferi-lo para o computador, representando-o em um formato utilizável pela máquina. As fontes de dados que apresentam conhecimentos sobre SGBD podem ser encontradas, em geral, em manuais técnicos, livros, artigos científicos, bases de dados, diagramas, entre outras. Já o conhecimento não documentado em SGBD reside na mente dos ABD que atuam no domínio executando suas atividades. Neste caso, a aquisição é por meio de interação direta.

O vocabulário utilizado pelo especialista para especificar um domínio com um não especialista é por muitas vezes inadequado para a solução de problemas, criando a necessidade de o engenheiro e o especialista trabalharem juntos para estendê-lo e refiná-lo.

Para isso serão demonstradas as técnicas de aquisição de conhecimento, e serão identificadas as formas como esse conhecimento pode ser adquirido (manual, semi-automático e automático) com foco na aquisição de conhecimento para a construção de ontologias.

Ontologias podem sofrer reuso ao logo de seu desenvolvimento e utilização. Na Etapa 4 será apresentado um exemplo de reutilização de classes de uma ontologia de domínio.

Manual

Martin e Oxman consideram que a aquisição manual de conhecimento é um processo iterativo com ciclos que dependem exclusivamente do tamanho do sistema a ser construído, levando em consideração a profundidade e amplitude das tarefas a serem suportadas, e a qualidade do conhecimento adquirido [Martin e Oxman 1988].

Neste processo manual as etapas são assim distribuídas: (1) Adquirir conhecimento inicial do domínio; (2) Criar um protótipo do conhecimento e testá-lo; (3) Dar ao protótipo um exemplo de tarefa; (4) Expor o sistema à observação do especialista; (5) Sair se a Base de Conhecimento (BC) estiver completa; senão avançar par a próxima etapa; (6) Inferir através de um especialista sobre o que é necessário para a BC; (7) Adquirir conhecimento indicado como necessário; (8) Adicionar o conhecimento necessário a BC; (9) Sair caso a BC estiver completa, caso contrario retornar à etapa 3.

Entretanto, a construção de uma ontologia a partir do zero é um processo lento, justamente pelas várias etapas que envolvem entrevistas com especialistas, compreensão do domínio do problema pelo engenheiro de conhecimento e formalização do modelo conceitual.

Além disso, a especificação manual de um grande número de documentos para compor uma ontologia pode se tornar uma atividade extremamente morosa, senão impraticável. Assim, quando possível, torna-se útil a construção de métodos automáticos ou semi-automáticos que realizem a construção da ontologia de forma mais dinâmica.

Semi-automática

Alguns trabalhos [Martins, 2006] e [Pampa, 2003] apresentam métodos de criação semi-automática de ontologias a partir do uso de textos de um domínio qualquer para a extração dos conceitos e relações presentes nesses textos. O trabalho de [Martins, 2006] apresenta uma alternativa para a AC manual baseada na comparação da freqüência relativa de termos extraídos com os escritos típicos da língua e na extração de padrões lingüísticos específicos. O método proposto foi compilado num mecanismo chamado *RulEE-SACT* que identifica termos candidatos a conceitos e relações existentes entre eles, apresentando-os a um ontologista para validação e, ao final, disponibiliza a ontologia ratificada para publicação e uso especificando-a na linguagem OWL (*Web Ontology Language*) [W3C, 2003].

Outras abordagens propostas (e.g. [Felicíssimo et al, 2003] , [Alani et al, 2003] e [Junior, 2008]) apresentam processos semelhantes para extração semi-automática tanto de conceitos quanto de restrições de integridade semântica capazes de servir como base para a concepção de ontologias de domínio.

Automática

Accioly [2002] apresenta em seu trabalho duas propostas de ambientes para aquisição de conhecimento automático baseadas em engenharia reversa que parte da informação armazenada em SGBD relacionais para dar origem a uma ontologia inicial que foram apontadas no início deste trabalho pela integração que mantêm com SGBD, sendo elas o *Karlsruhe Ontology and Semantic Web Infrastructure* (KAON) [Kaon 2001] e o *Domain Ontology Management Environment* (DOME) [O´Brien, 2000].

A aquisição automática representa um ganho potencial de tempo na definição semântica extraída de documentos para a construção de ontologias, porém depende da interação humana durante o processo de extração e composição dos termos ou então, no melhor dos cenários, depende dessa interação para validar os axiomas e relacionamentos gerados. Por isso, um dos grandes desafios da AC automática é buscar formas de converter automaticamente a fonte de dados em conhecimento, executando a interpretação e classificação das taxonomias.

Ferramentas automáticas só podem ser aplicadas quando existe uma fonte de informação compreensível pelo computador, ou seja, ela é limitada a fontes documentadas.

3.2.2. Etapa 2 – Conceitualização

Estruturar o domínio de conhecimento em um modelo conceitual é a tarefa principal desta etapa [Fernandez-Lopez, 1999]. Para isso, é importante compreender que a principal distinção entre ontologias e bases de conhecimento é que uma ontologia oferece uma estrutura que serve como base para a construção de uma base de conhecimentos em um nível superior. A ontologia é composta por um conjunto de conceitos e termos que descrevem um domínio, enquanto que a base de conhecimento utiliza essa estrutura e termos para descrever um ambiente. Quando esse ambiente é modificado, a base de conhecimentos também se modifica, entretanto, a ontologia não se altera até que o domínio seja alterado.

O uso de um modelo conceitual facilita a etapa da formalização do modelo conceitual, pois permite uma maior abstração do esquema representado pela linguagem UML, deixando de lado os detalhes técnicos de estruturação, e colocando o foco na construção de hierarquia de classes, propriedades e relacionamentos da ontologia de domínio.

3.2.3. Etapa 3 – Formalização

A etapa de formalização transforma o modelo conceitual em um modelo formal ou semi-computável [Fernandez-Lopez, 1999]. Portanto a definição de uma ontologia apresenta a existência de elementos que possuem categorias e significados que podem variar com relação à terminologia adotada pela área que os utiliza. Por isso, Gruber [1993] catalogou cinco componentes como fundamentais para a criação de uma ontologia independente do domínio a qual ela representa:

- **Classes ou conceitos:** expressam qualquer coisa sobre a qual pode existir uma forma de agrupá-las, como tarefas, funções, ações, estratégias entre outros. A Figura 3-1 apresenta um exemplo onde a classe Tarefa representa os tipos possíveis de tarefas: autônomas ou manuais;

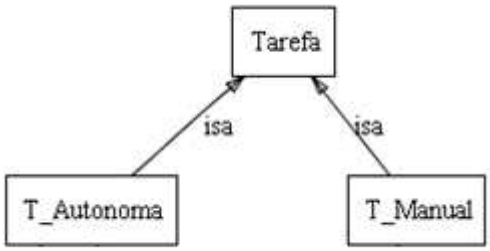

Figura 3-1: Hierarquia de classes

- **Relações:** simbolizam a interação entre as classes e seu domínio, ou seja, a forma como se conectam. Como exemplo, a figura 3-2 apresenta a utilização das expressões: "*executa*","*resolve*" e "*esta-associado*" como forma de manter relacionamentos não hierárquicos entre as classes;

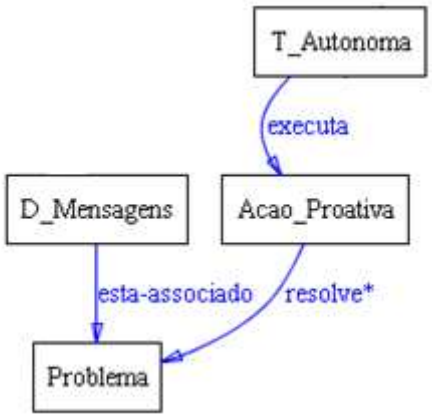

Figura 3-2: Relacionamentos entre as classes

- **Funções:** Uma função determina a forma como os relacionamentos serão quantificados. Tais funções são baseadas em quantificadores existências e universais baseados nos axiomas e em expressões matemáticas. Atualmente as ferramentas de desenvolvimento de ontologias se baseiam na criação de axiomas e descrições lógicas. O próximo item apresenta dois axiomas que podem dar origem a tais expressões matemáticas;
- **Axiomas:** São compostos por uma sentença, uma proposição, um enunciado ou uma regra que permita a construção de um sistema formal e se classificam em estruturais e não estruturais. A Figura 3-3 apresenta um axioma que indica que a classe D_Mensagens está associada somente a dois tipos de falhas: falha de sistema ou falha de transação. Esse tipo de axioma é responsável por criar os relacionamentos não hierárquicos entre as classes.

```
esta-associado SOME (Falha_de_Sistema OR
                     Falha_de_Transacao)
```

Figura 3-3: Axioma de associação entre as classes da ontologia

- **Instâncias:** representam elementos de uma ontologia, As instâncias são os conceitos e relações que foram criados na ontologia. Na Figura 3-4 São demonstradas duas instancias da classe D_Mensagens;

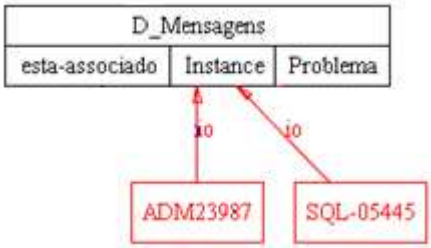

Figura 3-4: Instâncias de uma ontologia

A criação de ontologias conta com o apoio de ferramentas como: OntoLíngua [Farquhar et al, 1996], IKARUS [Skuce, 1996], WebOnto [Domingue, 2001], OntoEdit [Maedche et al, 2000], Protégé [Noy et al, 2000] entre outras. Essas ferramentas podem facilitar o processo de definição das classes, relacionamentos e axiomas, por ser tratar de uma tarefa dispendiosa. A sessão seguinte apresenta a implementação de uma ontologia.

3.2.4. Etapa 4 – Implementação

Dean e colegas [Dean et al., 2003] observando a necessidade de representar o conhecimento na área de Inteligência Artificial criou inicialmente uma linguagem baseada em HTML (chamada SHOE). Mais tarde, surgiu a linguagem baseada em XML (chamada XOL que posteriormente se tornou OIL), além de várias outras linguagens baseadas em *frames* e abordagens de aquisição de conhecimento como a RDF (Resource Description Framework) [Lassila, 1999], a OIL (*Ontology Inference Layer* ou *Ontology Interchange Language*) [Fensel et al 2001], a DAML+OIL (DARPA agent markup language) [Horrocks et al 2001] e finalmente a OWL (*Web Ontology Language*) [Dean et al, 2003].

Da mesma forma como foram apresentadas as metodologias de interesse desse trabalho e selecionada uma que atendesse às necessidades de criação do protótipo, foram identificadas três linguagens de desenvolvimento de ontologias e indicados os pontos positivos e negativos que fizeram com que a OWL fosse determinada como a mais indicada para o desenvolvimento.

Leva-se em consideração também o fato de que a OWL é atualmente um padrão utilizado quase na totalidade das ontologias desenvolvidas. O Quadro 3-2 apresenta essa coleta de dados acerca das linguagens para ontologias.

Quadro 3-2: Linguagens para ontologias

Linguagem	Descrição	Ponto positivo	Ponto negativo
RDF	Desenvolvida com o objetivo de representar o conhecimento através de redes semânticas. [Lassila 1999]	Possui uma semântica formal que utiliza o vocabulário baseado em conceitos localizados em URIs (RDF-S, que contêm os conceitos de classe, subclasse...) e uma sintaxe baseada em XML.	É uma linguagem não muito expressiva, permitindo apenas a representação de conceitos, taxonomias de conceitos e relações binárias.
DAML+OIL	DAML+OIL é uma linguagem de marcação semântica para a Web que apresenta extensões a linguagens como o DAML (DARPA Agent Markup Language). [Horrocks et al. 2001].	Mais ampla que RDF, permite representar conceitos, taxonomias, relações binárias e instâncias. É de muita expressividade.	Sua usabilidade foi bastante reduzida em virtude da criação da linguagem OWL que mantém suas características e incorpora novas funcionalidades.
OWL	OWL foi desenvolvida para ser utilizada por aplicações que processam o conteúdo da informação ao invés de apenas apresentá-la aos humanos. [Dean et al 2003]	Possui maior capacidade de interpretação de conteúdo da *Web* por máquinas em relação à linguagens como XML, RDF e RDFS.	São necessários ambientes gráficos de desenvolvimento como o OWLViz ou extensão das funcionalidades de edição deste *plug-in*

Dean [Dean et al, 2003] afirma que a linguagem OWL pode ser representada de três formas:

- **OWL *Full*:** essa sub-linguagem permite ampliar o vocabulário pré-definido de uma ontologia escrita em RDF ou OWL. Porém é dificilmente compatível com softwares de inferência já que estes não conseguem suportar completamente os recursos da OWL *Full*, ou seja, enquanto a OWL DL impõe restrições sobre o uso de RDF a OWL *Full* permite misturar OWL com RDF tornando possível, por exemplo, que uma classe seja ao mesmo tempo uma classe e um indivíduo;

- **OWL DL:** serve para garantir o conceito de computabilidade (todas as conclusões são computáveis) e decidibilidade (todas as computações tem um tempo pré-determinado para se encerrar). A sigla DL se refere a lógica descritiva (*description logics*), e determina sobretudo que sua estrutura de restrições seja mais complexa que a sub-linguagem *Lite* definindo por exemplo que uma classe possa ser subclasse de muitas classes, sem que ela seja instância de outra classe;

- **OWL *Lite*:** é uma sub-linguagen da OWL DL que usa somente algumas características da linguagem OWL e, por isso, é mais limitada que OWL DL ou OWL *Full.* Ela é em tese utilizada em aplicações que necessitam simplesmente de uma classificação hierárquica e restrições simples já que as restrições suportadas por ela são apenas as referentes à cardinalidade 0 ou 1*;*

A escolha pela linguagem OWL-DL é pautada no fato de que ferramentas de desenvolvimento de ontologias, como o Protégé [Noy et al., 2000] (com o *Plugin* OWL), permitam aos usuários, além de desenvolver ontologias, encontrar erros e detectar inconsistências. O fato de que essa ferramenta é capaz de gerar classes correspondentes em JAVA a partir de uma ontologia em OWL foi levado em consideração, assim como o fato da linguagem OWL ser recomendada como linguagem padrão para desenvolvimento de ontologias segundo o W3C[1] *(World Wide Web Consortium).*

[1] http://www.w3.org

Esta sessão apresentou uma metodologia e suas etapas necessárias à concepção de uma ontologia de domínio. A sessão seguinte apresenta uma metodologia de criação de sistemas

3.3 SADAAM (*Software Agent Development An Agile Methodology*)

Combinar Engenharia de Software orientada a agentes (AOSE) e técnicas ágeis, é a proposta da metodologia SADAAM (*Software Agent Development an Agile Methodology*) que visa tornar o ciclo de desenvolvimento de software mais flexível, criando assim meios de se desenvolver softwares orientados a agentes mais interativos, incrementais e com ênfase na fase de testes. É produto recente de pesquisa e prima, sobretudo, pelo desenvolvimento de software seguindo a metodologia de desenvolvimento incremental, embora apresente pouca importância à fase de levantamento de requisitos [Clynch, 2007].

Essa metodologia é baseada em cinco fases, se consideradas duas fases externas ao processo incremental de desenvolvimento (a fase inicial de especificação de requisitos e a fase final de término de projeto). As outras três fases principais são indicadas na Figura 3-5, seguem o fluxo advindo dos dados levantados na fase de especificação de requisitos, e são respectivamente: **(a)** Projeto (*Design*); **(b)** Implementação baseada em testes (*Test driven implementation*) e **(c)** Liberação e revisão (*Release and review*).

Após a execução da etapa responsável pela liberação e revisão do sistema, entra em ação o processo de Refatoração e aprimoramento (*Refactor and enhancement*) cuja função é tornar o desenvolvimento incremental, ou seja, é através desse processo que o desenvolvimento do software retorna à fase pós-levantamento de requisitos para novas implementações até que sejam testadas e novas funcionalidades sejam implementadas ou que o software esteja concluído.

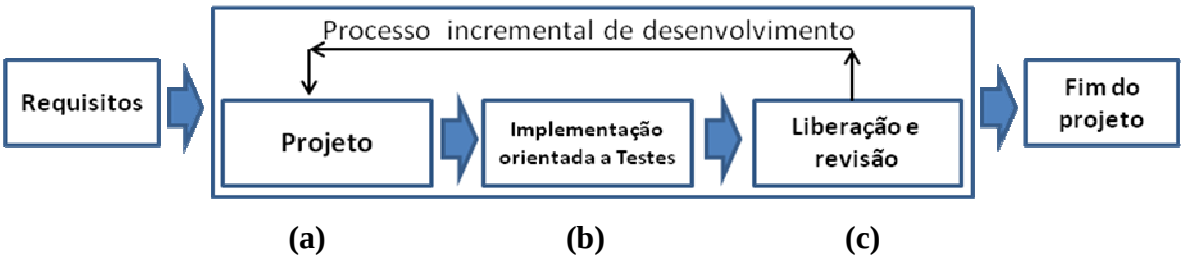

Figura 3-5: Arquitetura da metodologia SADAAM adaptado de [Clynch, 2007]

3.3.1. Projeto

Uma peculiaridade que é comum a algumas metodologias é a de não possuir recursos avançados ou sofisticados de especificação de requisitos. Por isso, elas se apóiam em outras metodologias para essa fase, assim como ocorre na fase de design, onde se utiliza a linguagem *Agents Unified Model Language* (AUML) [AUML, 2008] para definir os casos de uso e as classes do sistema a ser modelado como é o caso da metodologia GAIA [Zambonelli et al. 2003] e mesmo essa metodologia aqui apresentada, que utiliza AUML em sua fase de Design para apresentar o diagrama de interação entre os agentes e outros diagramas.

A fase de design possui como finalidade elicitar exigências e necessidades do sistema multiagentes a ser desenvolvido e traduzi-los para um projeto de software. Sua origem é a fase anterior, onde são definidos os requisitos. Segundo Clynch [Clynch, 2007] é nesta fase que utilizando o AUML [AUML, 2008] é definido, detalhado e documentado o projeto de software. Para tal, são criados os seguintes artefatos: (a) Comportamento do sistema; (b) Papéis executados por agentes; (c) Relacionamentos, interações, e atividades e (d) Classes e os métodos dos agentes. A Figura 3-6 apresenta as etapas que compõem essa fase e as próximas seções apresentam os detalhes dessas definições e a criação dos diagramas necessários.

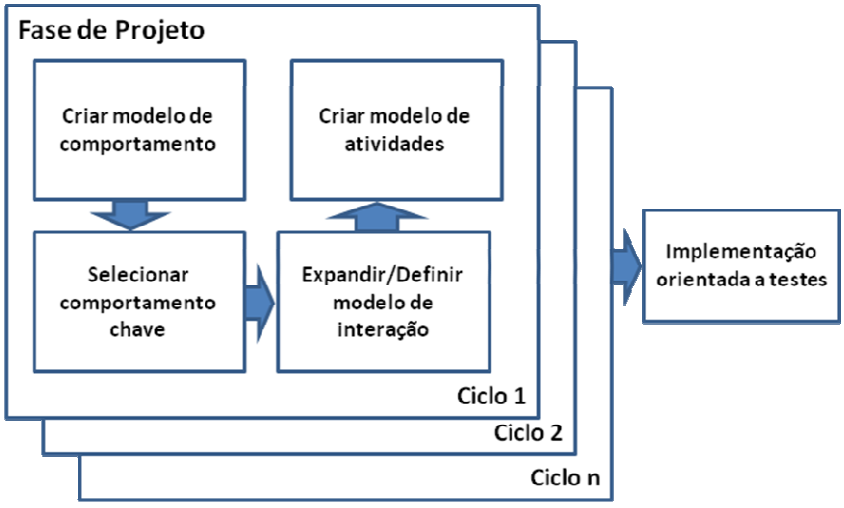

Figura 3-6: Etapas da fase de Design, adaptado de [Clynch, 2007]

Criação do modelo de comportamento do sistema

Nesta fase é criado o modelo de comportamento do sistema (*System Behaviour Model* - SBM) para identificar os comportamentos chave do sistema de acordo com o ambiente especificado. O comportamento do sistema é composto pelas atividades e interações que fazem parte dos processos de operação do sistema. A representação destas atividades e

interações é feita através de diagramas AUML [AUML, 2008] e modificados de forma a atender a necessidade de modelar tanto a estrutura organizacional quanto o comportamento dos agentes. As figuras a seguir apresentam diagramas baseados em um modelo conceitual de uma rede de abastecimento, composta por vendedores, compradores e transportadores. (Figuras 3-7 e 3-8).

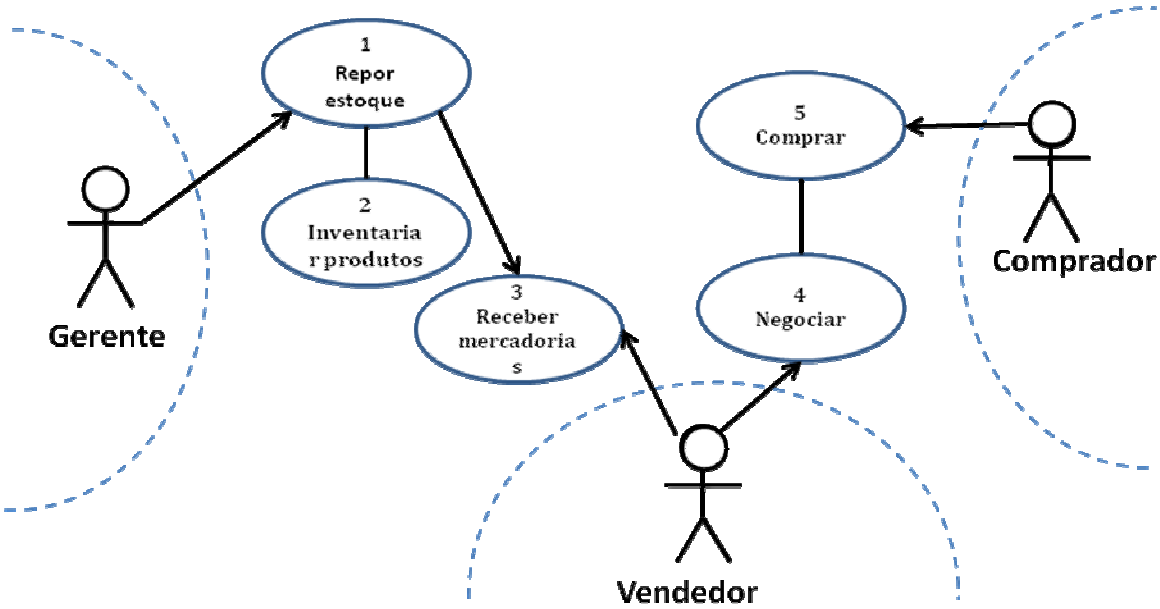

Figura 3-7: Diagrama de comportamento do sistema

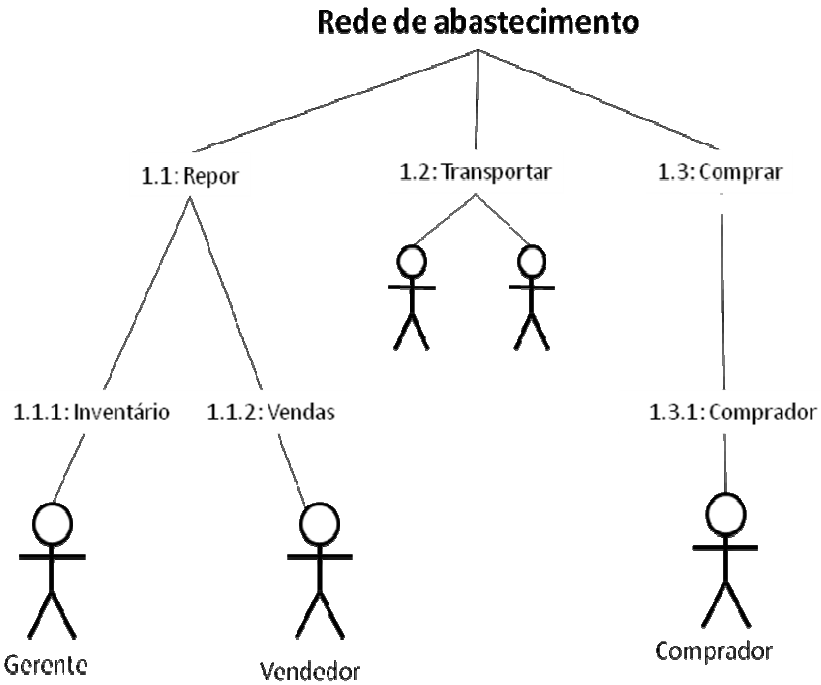

Figura 3-8: Diagrama de unidade organizacional, adaptado de [Clynch, 2007]

Selecionar comportamentos chave do sistema

Esta etapa determina um dos sub-conjuntos entre os comportamentos chave apresentados na etapa anterior para que seja feia uma análise mais aprofundada. É nessa etapa que se identificam as partes do sistema em que a equipe de desenvolvimento vai trabalhar de forma mais efetiva no desenvolvimento dos códigos fontes.

Na Figura 3-8 foram apresentados os comportamentos de alguns agentes como: "comprador", "vendedor" e "gerente". Cabe a essa etapa definir cada um desses comportamentos de forma pormenorizada. (eg.: Criar o diagrama de comportamento do comportamento 1: *Repor estoque.*

Expandir/Definir modelo de interação

O modelo da interação (*Interaction Model - IM*) define as interações que ocorrem entre agentes durante um comportamento específico do sistema. Este modelo utiliza o padrão de diagramas de interação do protocolo UML. A Figura 3-9 apresenta o cenário de interação entre os agentes vendedor, o agente comprador e o gerente do armazém.

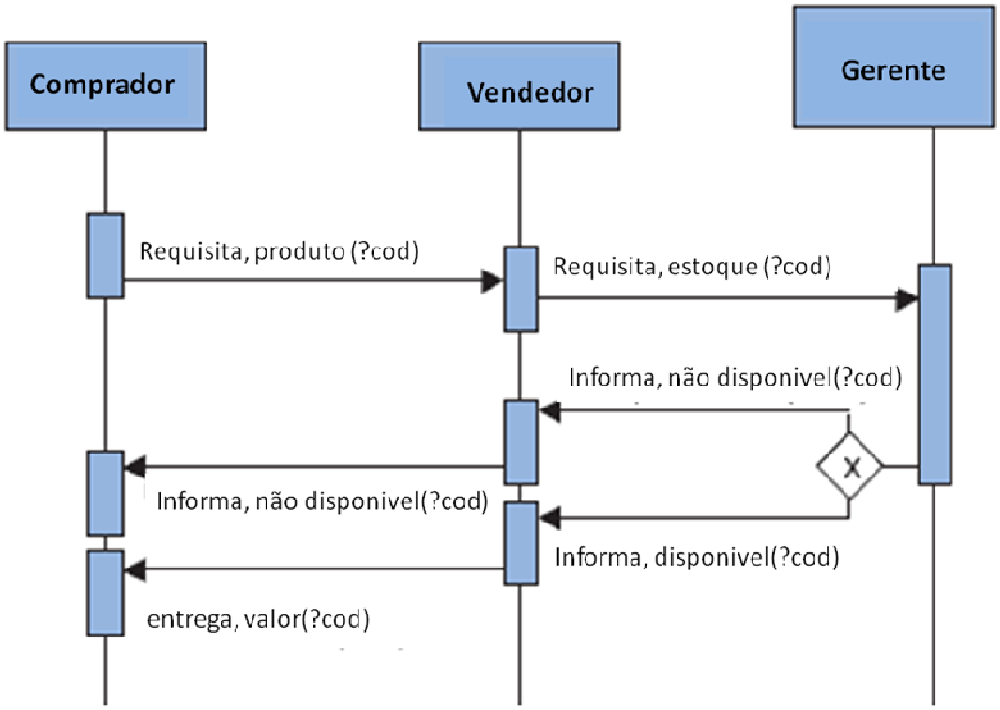

Figura 3-9: Diagrama de interação entre os agentes adaptado de [Clynch, 2007]

Expandir o Modelo de Interação (IM) usando o Modelo de Atividades (AM)

Nesta etapa, é criado o modelo da atividade (*Activity model - AM*) que identifica o conjunto de atividades realizadas por agentes no sistema. Este modelo emprega os diagramas customizados do formato UML onde: as caixas representam atividades individuais; as associações os requisitos e desempenho das atividades; e os traços abaixo dos agentes representam sua associação às respectivas atividades.

Esta etapa é responsável por estender o modelo de interação entre agentes revelando assim o maior número possível de ações dos agentes, ou seja, aquelas que no diagrama de interação possuem dois ou mais fluxos paralelos.

A Figura 3-10 mostra o diagrama de atividades para o exemplo apresentado no diagrama de interação entre os agentes (Comprador, Vendedor e Gerente).

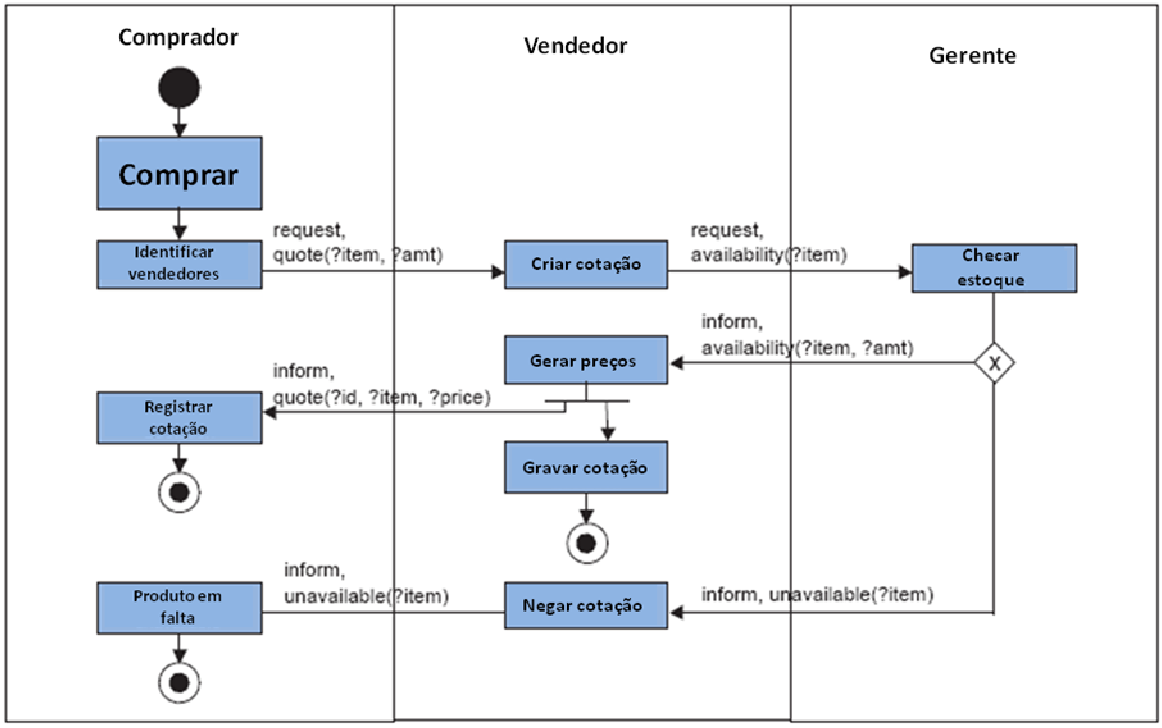

Figura 3-10: Diagrama de atividades adaptado de [Clynch, 2007]

A próxima fase é responsável pela implementação do projeto utilizando orientação baseada em testes e complementa a fase do Design apresentada nesta seção.

3.3.2. Implementação orientada a testes

A fase de Implementação Orientada a Testes (*Test-Driven Implementation)* emprega técnicas ágeis para apoiar a criação, testes e execução dos agentes. Esta etapa prioriza os testes com os agentes de software de uma maneira diferente das implementações desse tipo, pois essa metodologia prega a criação de agentes de teste baseados nos agentes reais de forma que ambos interajam no mesmo ambiente.

A Figura 3-11 apresenta uma visão deste cenário de criação das cópias dos agentes e divide a arquitetura desta etapa em três conjuntos, sendo eles: Fase de implementação baseada em testes, estrutura do conjunto de testes da aplicação e plataforma de teste e integração de agentes.

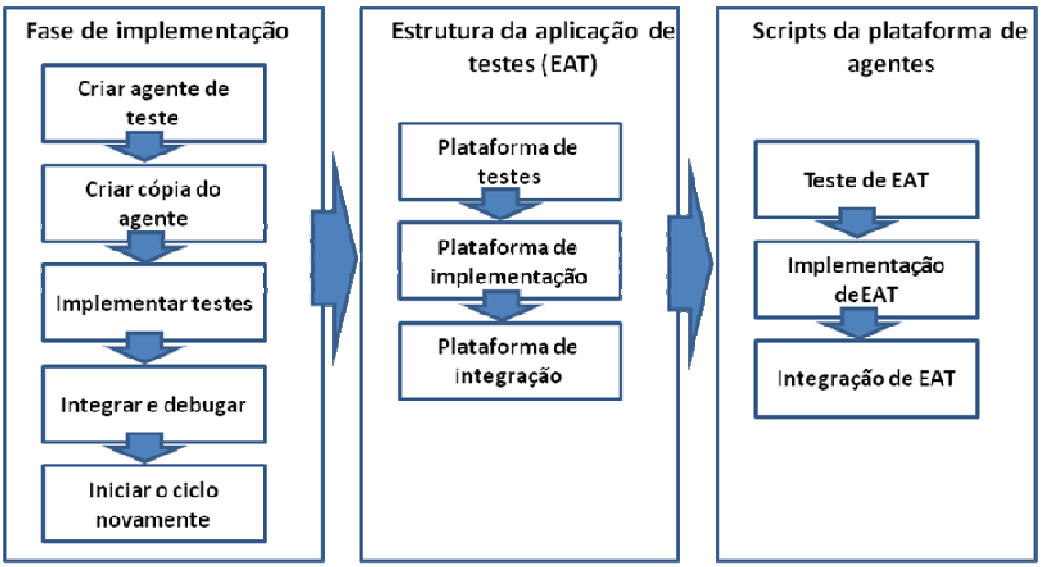

Figura 3-11: Estrutura da fase de implementação baseada em testes [Clynch, 2007]

O funcionamento desta etapa consiste nas seguintes tarefas:

(a) Criar um agente de testes: O agente de testes é um agente criado para ser inserido no ambiente da aplicação para verificar seus comportamentos;

(b) Criar uma aplicação que simula um agente: corresponde a criar um clone *(AAUT - Application Agents Under Test)* do agente de testes e verificar se este agente mantém a capacidade de execução de tarefas com pequenas mudanças comportamentais. Podem-se exigir muitas situações de teste até que seja satisfeita alguma condição de teste para esse novo comportamento;

(c) Implementar testes e executar: consiste em executar os testes com os agentes de testes e os AAUT e verificar se eles precisam de correções;

(d) Integrar e depurar AAUT: integrar e depurar os códigos fonte dos agentes AAUT se necessário até que seu comportamento esteja adequado ou correto;

(e) Iniciar o ciclo novamente: como o processo de desenvolvimento e testes é incremental, quantas vezes forem necessárias, será preciso reiniciar o sistema em busca de falhas.

A metodologia SADAAM defende a criação do ambiente de testes baseado na linguagem AFAPL2 [AgentFactory, 2008] que suporta a execução de agentes intencionais. Os agentes intencionais são uma classe de agentes que raciocinam sobre o que fazer através de um jogo de modelos mentais tais como a opinião, os objetivos, as obrigações, os compromissos e as intenções. Essa metodologia defende o uso desta linguagem porque, além de propor um modelo de trabalho, também traz o conjunto completo de ferramentas de criação e testes desses agentes. A principal vantagem em se utilizar essa linguagem é que se pode seguir o modelo de trabalho proposto pela metodologia e ainda utilizar as ferramentas desenvolvidas para ela. Por outro lado, a mesma se restringe o desenvolvimento de agentes à esta linguagem.

3.4 Padrão de comunicação entre agentes

Bloisi [Bloisi 2004] afirma que para realizar tarefas do quotidiano as pessoas em uma sociedade organizada se auxiliam de forma direta ou indireta para realizar as mais variadas atividades como comprar algo, transportar objetos ou realizar trabalhos. A comunicação entre as pessoas é a garantia de que a interação e cooperação vão assegurar que tais trabalhos sejam realizados e fornece o resultado, se foi executado com sucesso ou não.

Os agentes inteligentes de software agem de maneira similar aos humanos no que tange a necessidade de comunicar-se e interagir. Tais situações são encontradas quando os agentes trocam informações sobre o ambiente em que estão inseridos, respondem questões formuladas por outros agentes, compreendem solicitações feitas por outros agentes até mesmo quando compartilham as experiências adquiridas nesse ambiente.

Segundo Russell o que classifica um agente de software como um agente inteligente é a sua capacidade de reagir racionalmente aos estímulos do ambiente no qual ele está inserido,

e o grau de inteligência atribuído a este agente pode ser medido de acordo com a capacidade que ele possui de realização de ações [Russell e Norvig 1995].

A realização de ações e comportamentos também é apresentada por [Wooldridge 2002] como fator de determinação de inteligência em agentes de software. Segundo este trabalho, os comportamentos esperados de um agente são: (a) Autonomia, (b) Capacidade de comunicação, (c) Capacidade de cooperação, (d) Capacidade de raciocínio, (e) Planejamento e (f) Adaptabilidade.

Segundo Nwana [Nwana, 1996] a autonomia é representada pelos agentes que agem com base em suas regras de decisão sem a interferência de outros agentes ou usuários. Os agentes possuem estados e metas internas, e executam ações para atingir estas metas. A Figura 3-12 ilustra esses estados e metas.

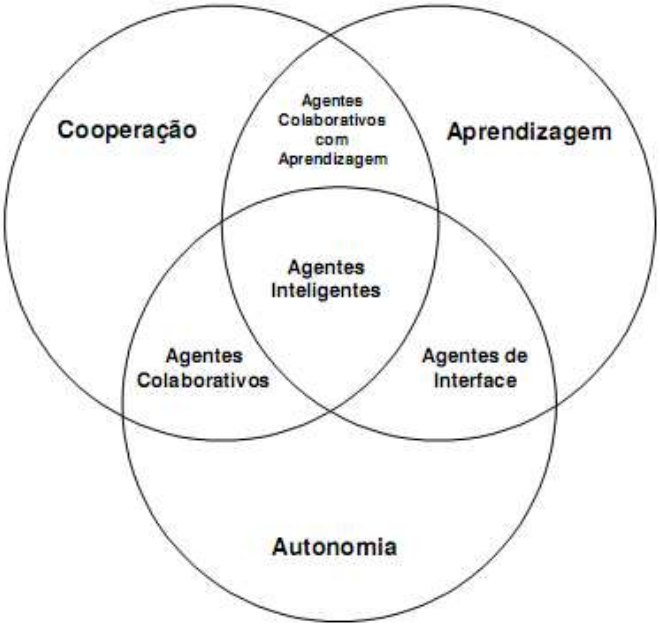

Figura 3-12: Categorias de classificação de agentes por Nwana [Nwana, 1996]

Seguindo essa classificação, quanto maior o nível de autonomia, mais ampla se tornará a capacidade dos agentes em executar ações periódicas, espontâneas e tomar iniciativas próprias. Isso torna os agentes mais pró-ativos, que são características altamente desejadas em qualquer sistema autônomo. Na seção seguinte será descrito o modelo e a tecnologia que será utilizado nesse trabalho para a comunicação entre agentes e sua utilização da ontologia de domínio.

Modelos de comunicação entre agentes

Modelos de comunicação entre agentes são utilizados para definir a maneira pela qual os agentes irão se comunicar. Independente do modelo proposto, o sincronismo da troca de informações e o relacionamento entre os agentes são fatores a serem considerados durante a escolha de um modelo que possa ser aplicado a um projeto que utilize agentes autônomos. Atualmente os principais modelos de comunicação entre agentes são baseados na troca de mensagens ou na utilização de quadro-negro (*blackboard*).

Lemos [2002] faz analogia do quadro negro à uma memória compartilhada, onde os agentes de software podem ler e escrever, compartilhando assim as respostas com os agentes que necessitem executar alguma tarefa enquanto outros agentes anotam suas perguntas a espera de novas respostas não encontradas nesse quadro.

Enquanto o modelo baseado em quadro-negro consiste em utilizar um mecanismo comum aos agentes embora esses não necessariamente necessitem manter comunicação para obter as respostas, outro modelo adotado para comunicação entre agentes consiste na comunicação direta entre eles através da troca de mensagens.

Helin [2002] apregoa um modelo de comunicação que se baseia no conceito de que cada agente conheça a identificação e localização dos agentes inseridos no sistema para que possa ser criado um fluxo de troca de mensagens e esse conjunto de agentes conta ainda com agentes responsáveis por armazenar endereços e serviços dos agentes atuadores no sistema.

Nesse modelo, pode haver dois cenários: (a) Os agentes localizam diretamente outros agentes ou serviços que estejam sendo indicados como disponíveis em locais de armazenamento chamados de *yellow pages* (serviços disponíveis) ou *white pages* (agentes disponíveis); (b) Os agentes solicitam que um agente facilitador faça essa busca e lhe retorne uma mensagem de resposta com os dados solicitados.

Ambos os cenários são demonstrados através da Figura 3-13. Esse tipo de linguagem de comunicação entre agentes (*Agents communication language*) ACL precisa obedecer a uma sintaxe pré-definida que estabeleça um protocolo de envio, recebimento, compreensão e resposta para essas mensagens.

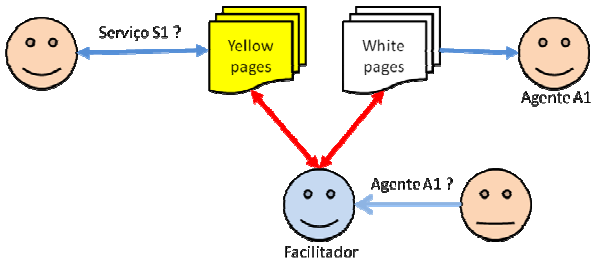

Figura 3-13: Agentes podem utilizar um facilitador [Lemos et al., 2002]

A seção seguinte trata de uma linguagem especificada para manter a comunicação entre agentes.

FIPA ACL

A linguagem FIPA-ACL [FIPA, 2008] contém vinte e dois (22) tipos de mensagens organizadas em quatro (4) categorias criadas pela FIPA, sendo elas: **(a)** Transferência de informação; **(b)** Negociação; **(c)** Ação e **(d)** Gerenciamento de erros. A cada um desses tipos de mensagens é dado o nome de performativa. As principais performativas usadas nesse trabalho são detalhadas através de exemplos no Capitulo 5. Nessa seção serão apresentados dois exemplos de troca de informações entre agentes, utilizando as performativas mais comumente utilizadas: ***inform*** e ***query-ref.*** No Quadro 3-3 é apresentado um caso simulado de comunicação entre agentes onde um agente 1 (A1) envia uma mensagem usando a performativa **inform** ao agente 2 (A2) transmitindo a ele uma mensagem de aviso.

Quadro 3-3: Agente 1 envia mensagem ao Agente 2

```
01( inform
02          :sender     A1
03          :receiver   A2
04          :content    (aviso)
05          :reply-with idComm0001
06          :language   FIPA-sl
07          :ontology   Ontology-X
08 )
```

Em seguida, o agente A2 pesquisa na ontologia e localiza uma resposta para a situação informada pelo agente A1 e envia como resposta a esse agente, através do uso da performativa query-ref, uma mensagem contendo uma instrução para corrigir determinado

problema, porém ao consultar as páginas amarelas (*Yellowpages*) indica na mensagem que o agente 3 (A3) é quem deve proceder a execução desse serviço (Quadro 3-4).

Quadro 3-4: Agente 2 responde ao Agente 1

```
01( query-ref
02          :sender      A2
03          :receiver    A1
04          :content     (comando)
05          :in-reply-to idComm0001
06          :reply-with  idComm0002
07          :language    FIPA-sl
08          :ontology    Ontology-X
09    )
```

Portanto, de acordo com os modelos aqui apresentados é possível verificar que esse padrão de comunicação prioriza a utilização da linguagem FIPA-ACL [FIPA 2008] por se tratar de uma linguagem especificada por uma organização ligada a IEEE [IEEE, 2008] cujo interesse de padronização e interoperabilidade com outras tecnologias e padrões vai ao encontro da idéia aqui proposta.

3.5 Considerações

O objetivo deste capítulo foi descrever duas metodologias e um padrão necessários ao desenvolvimento deste trabalho. Respectivamente, foram apresentadas a metodologia Methontology, a metodologia SADAAM e o padrão FIPA-ACL.

A metodologia para desenvolvimento de ontologias de domínio Methontology foi apresentada de forma detalhada, comentando-se cada uma de suas etapas e quais as atividades de cada uma delas. Além disso, foi indicada como a metodologia escolhida para desenvolver a ontologia de domínio proposta nesse trabalho por apresentar integração com a plataforma tecnológica escolhida, atender aos padrões da FIPA em relação a sua linguagem utilizada (OWL) e por ser incremental.

A metodologia apresentada para o desenvolvimento de sistemas multiagentes é relativamente recente e assim como as demais desse contexto apresenta pontos fortes e fracos. O que chamou a atenção para a sua utilização e escolha é a capacidade de expressar os comportamentos do sistema a ser modelado bem como a riqueza que os diagramas fornecem

ao detalhar comportamentos de agentes de software e a interação que eles mantêm com outros agentes.

O capítulo foi finalizado com a apresentação do padrão de comunicação entre agentes de software FIPA-ACL que teve sua escolha para esse trabalho baseada no quesito padronização. Pois a utilização deste padrão de comunicação permite o desenvolvimento de rotinas de comunicação entre agentes integradas às demais tecnologias que serão apresentadas no Capítulo 4 quando será tratada a especificação do OntoSitter.

Capítulo 4
Definição da Ontologia de Domínio para SAGBD

Este capítulo detalha a concepção da ontologia de domínio de problemas em sistemas autônomos de gerenciamento de bancos de dados (SAGBD) utilizada pelo sistema OntoSitter. Essa ontologia foi definida com a utilização da metodologia Methontology [Lopez, 1997].

4.1. Introdução

Atualmente existem na literatura várias metodologias para o desenvolvimento e definição de ontologias, dentre as quais se destacam a CYC [Lenat, 1990], Maedche [Maedche, 2000] e Methontology [López, 1997]. Essas metodologias foram apresentadas no capitulo 3 desse trabalho.

A metodologia proposta por López apresenta várias etapas, das quais foram priorizadas: *especificação*, *conceitualização*, *formalização*, *integração* e *implementação*. A clareza na definição das etapas e os artefatos gerados por essas etapas, também serviu como balizador para o desenvolvimento da ontologia de domínio de SGBD com essa metodologia.

Este capítulo trata a concepção de uma ontologia de domínio de problemas em SAGBD e a definição semântica do relacionamento entre os problemas, falhas e ações para corrigi-los ou impedi-los. Para isso, a Seção 4.2 apresenta a definição da ontologia de domínio e cada uma de suas etapas anteriormente citadas, e a Seção 4.3 encerra esse capítulo com as considerações sobre a concepção da ontologia.

4.2. Definição da ontologia de domínio

A definição de uma ontologia de domínio para categorizar problemas e suas soluções em SGBD segue as etapas da metodologia Methontology [Lopez, 1997] e visa apresentar um modelo que seja passível de reuso, tanto para a classificação de problemas em SAGBD quanto para aplicações que façam uso desta ontologia para interação com outros sistemas.

4.2.1. Especificação

A etapa de especificação define, segundo [López, 1997], o porquê de a ontologia estar sendo construída e apresenta na aquisição de conhecimento (AC) o principal recurso para sua especificação. Por isso, foi necessário definir a forma como o mapeamento de erros, termos e rotinas foram adquiridos.

No domínio de erros em SAGBD e rotinas executadas tanto pelo ABD quanto por agentes de software, é levado em consideração o fato de que, cada SGBD fornece um conjunto de mensagens capaz de identificar os eventos recorrentes em suas rotinas. Em alguns casos, o volume de mensagens pode ser bastante grande e evoluir continuamente a cada nova interação. [Gabriel, 2000]. Por exemplo, o SGBD Oracle [OracleADM, 2008] possui um conjunto de mais de 1800 mensagens que representam avisos e identificam erros. Apesar de

numeroso, esse conjunto revela poucas mensagens contendo informações precisas sobre a solução para problemas e a ontologia torna essas mensagens mais completas, apresentando a solução através de comandos SQL.Por isso, a etapa de especificação desta ontologia se baseia no fato de que os atores ligados ao SAGBD (ABD e agentes) são diretamente influenciados pelas informações constantes em repositórios de eventos (parâmetros e mensagens) para que possam executar suas tarefas (reativas ou proativas).

Sendo assim, segundo [López, 1997], o artefato gerado por essa etapa consiste em justificar a construção de uma ontologia. A justificativa para definir a ontologia de domínio de problemas em SAGBD é dada pela necessidade de classificação semântica do relacionamento entre problemas e erros que ocasionam as falhas.

4.2.2. Conceitualização do domínio de um SAGBD

Esta seção apresenta a estruturação do domínio de conhecimento (neste caso, o domínio de problemas, erros e falhas de sistema) em um modelo conceitual de SAGBD. Esta etapa corresponde à etapa de conceitualização na Methontology [Lopez, 1997] apresentada no Capítulo 3. Nessa etapa é apresentado o contexto e as especificações da própria metodologia, o que pode ser subtendido como a etapa responsável por mapear a especificação da ontologia em diagramas que representem o contexto no qual ela está inserida. A Figura 4-1 ilustra o contexto mencionado anteriormente, que detalha esses relacionamentos entre agentes de software e o ABD pertencentes ao domínio especifico dos SAGBD.

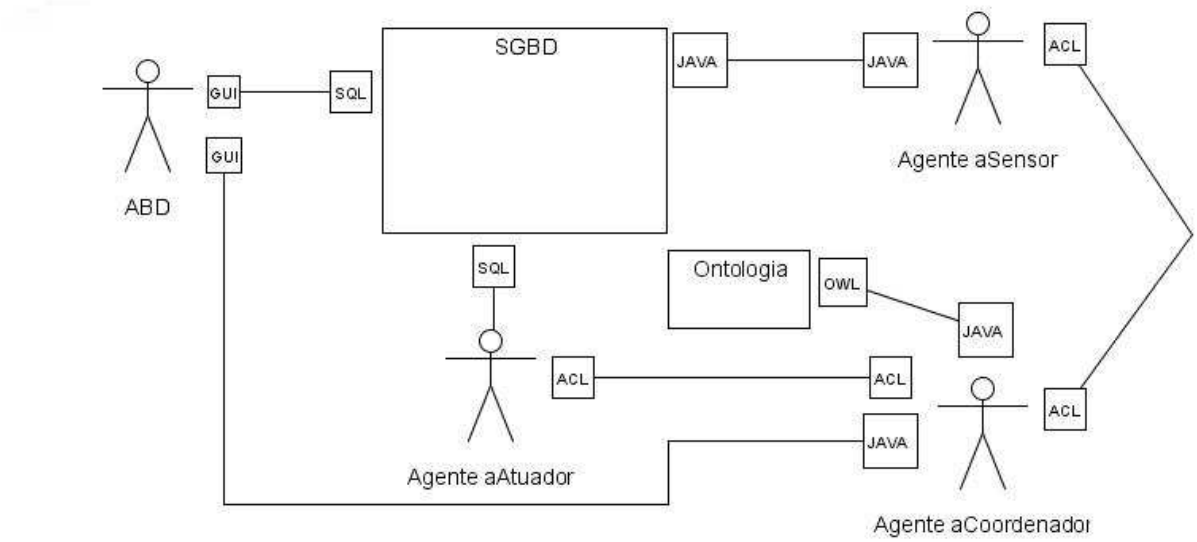

Figura 4-1: Contexto para um SAGBD

Após a construção de um diagrama ou representação do domínio de contexto do SAGBD, a criação de uma hierarquia de classes é um fator importante para a compreensão do relacionamento que esses recursos mantêm em relação a um sistema autônomo. A representação gráfica dessa ontologia de domínio e suas relações de hierarquia são mostradas na Figura 4-2.

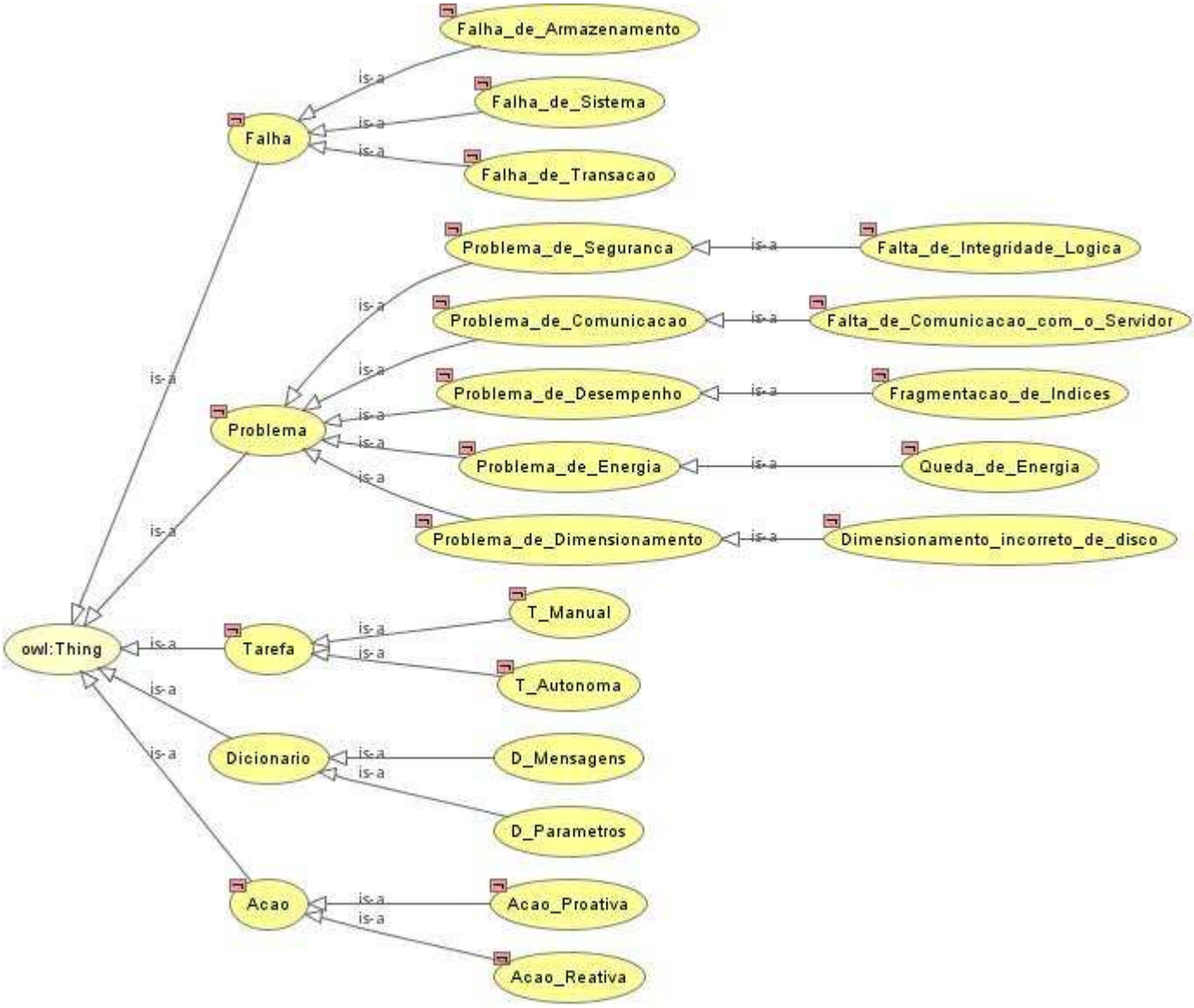

Figura 4-2: Hierarquia de classes da ontologia para um SAGBD

Para esse trabalho, compreende-se que uma ontologia é composta por relacionamentos hierárquicos e não hierárquicos. Um relacionamento hierárquico é do tipo ***"é-um" (is-a)***, e pode ser representado através de setas entre as classes, assim como indicado na Figura 4-2.

Um relacionamento dito não hierárquico é representado pelas propriedades que criam elos entre as classes em uma ontologia. Um exemplo desse tipo de relacionamento pode ser observado através da Figura 4-3, onde a classe **Acao_Reativa** mantém um relacionamento não hierárquico com a classe **Falha** baseado na propriedade **"resolve"**.

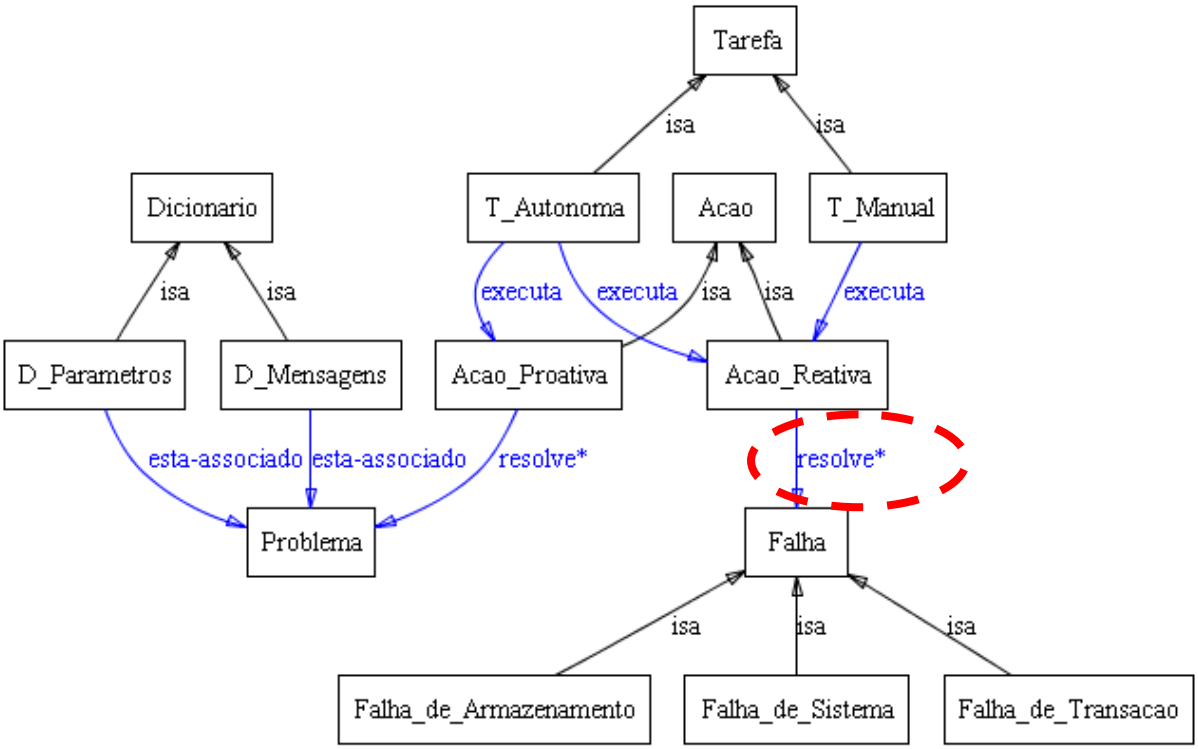

Figura 4-3: Relacionamentos hierárquicos e não hierárquico entre classes

A figura anterior contempla as principais classes desta ontologia: Tarefa, Problema, Falha, e Ação. Em nenhum momento existe um relacionamento hierárquico direto entre essas classes, porém os relacionamentos não hierárquicos criam uma rede de relacionamentos que conectam todos esses conceitos.

É possível por exemplo verificar que somente ações reativas são capazes de resolver falhas, enquanto as ações proativas são voltadas à resolução dos problemas, e com isso, é apresentada a diferença entre tais classes na forma como são catalogadas.

A etapa seguinte é responsável pela formalização dessas classes e pelo detalhamento de cada uma das classes pertencentes a essa ontologia de domínio.

4.2.3. : Formalização das classes da ontologia

A criação da ontologia de domínio de um SAGBD foi formalizada com o uso da ferramenta Protégé [Protégé, 2008] e foi dividida em 5 classes principais, as superclasses, que são: "**Falha**", "**Acao**", "**Dicionario**", "**Problema**" e "**Tarefa**".

Na área de Engenharia de Software, os termos: problema, erro e falha são definidos de forma diferente, embora sejam o erro e a falha derivados de situações problemáticas que alterem ou impeçam de se atingir o objetivo para a qual determinada função foi desenvolvida. [Rausand e Oien, 1996].

Ainda segundo Rausand e Oien, o termo "falha" é freqüentemente confundido com os termos "falta" e "erro", existindo várias definições para estes termos. De acordo com o [IEC 50, 2008] *International Electrotechnical Commission*, que é o órgão responsável por criar o vocabulário internacional eletrotécnico, um erro é representado pela discrepância entre um valor (ou condição) medido, observado ou computado, e a verdade (especificada ou teoricamente correta). Assim, é correto afirmar que um erro não necessariamente é uma falha pelo fato de estar dentro de limites aceitáveis de desvio de um desempenho desejado (valor alvo).

Sendo assim, uma situação originária de problema é determinante para que ocorram erros (independente do fator desempenho) e esses por sua vez podem ao longo do tempo causar falhas como é indicado na Figura 4-4.

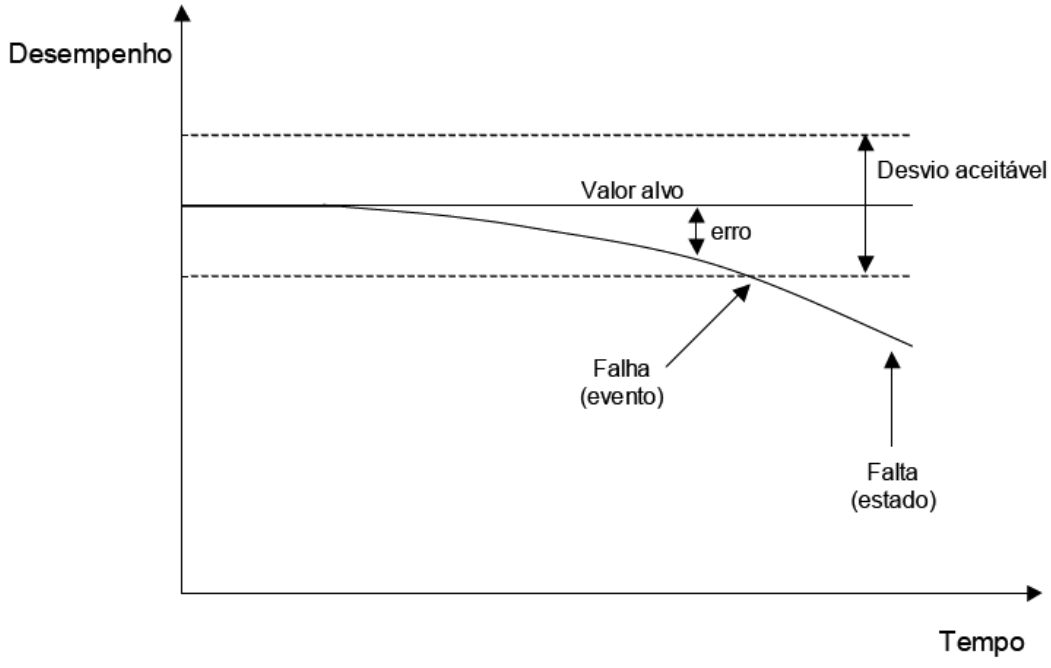

Figura 4-4: Representação gráfica para os termos problema, erro e falha segundo [Rausand e Oien, 1996]

Com base nessa definição, esta ontologia foi desenvolvida de forma a apresentar as classes **Problema** e **Falha** como determinantes para o processo de compreensão dos eventos gerados nos SGBD. O conceito de erro servirá como classificação para determinar se o problema é recorrente ou ocasional no universo dos SAGBD e, para isso, não será instanciado como classe, apenas como uma propriedade transitiva. Aliado a essa compreensão, pode-se classificar que tipos de ações (reativas ou proativas) devem ser tomados diante destes eventos.

Problema

Essa classe representa um conjunto de situações que englobam motivos de geração de erros de funcionamento dos SGBD. Partindo do conceito de que um problema é a situação que pode ocasionar um erro e esse conseqüentemente uma falha, é lógico afirmar que as falhas nos sistemas poderiam ser evitadas caso os problemas fossem corrigidos ou contornados. Entretanto, caso não o seja, o problema dá origem a um erro e esse à falha, que só pode ser resolvida de forma reativa, uma vez que já aconteceu. O conceito de falha é tratado de forma diferente do problema, que pode ser resolvido na maioria dos casos de forma proativa por se tratar de uma situação provável de acontecer. A classe **Problema** possui relacionamento hierárquico com os diferentes tipos de problemas detectados em SGBD. A Figura 4-5 apresenta um exemplo de um problema relacionado à fragmentação de índices, que é classificado como um erro recorrente e que ao longo do tempo pode dar origem a uma falha por impedir que o sistema execute rotinas que utilizem tais índices.

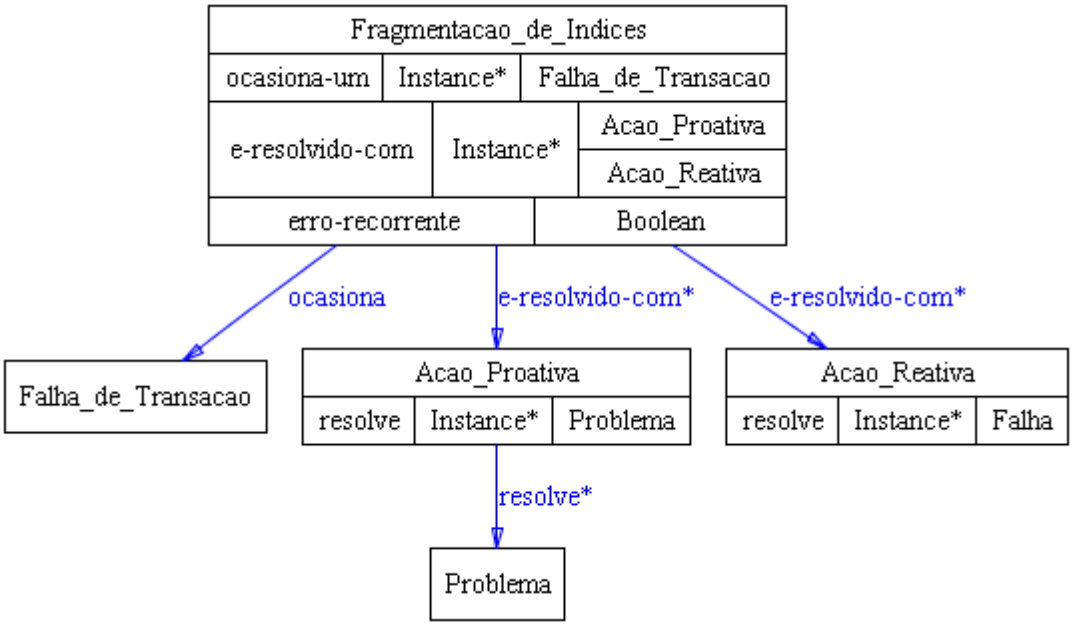

Figura 4-5: Exemplo de um problema em SGBD

Falha

A ontologia de domínio apresenta a classe **Falha** como uma classe que depende dos relacionamentos não hierárquicos com a classe **Problema**. Conforme demonstrado anteriormente, um erro é ocasionado por situações que representem problemas ao funcionamento dos sistemas. Para isso, a classe **Falha** foi dividida em três classes disjuntas: **Falha_de_Sistema**, **Falha_de_Transacao** e **Falha_de_Armazenamento** cujo objetivo único é cobrir o maior número possível de falhas oriundas de erros em SGBD.

A Figura 4-6 apresenta as três classificações para Falhas, e como exemplo, apresenta um problema que é classificado como um erro recorrente, a ocorrência deste tipo de erro ocasiona um tipo específico de falha: a falha de armazenamento.

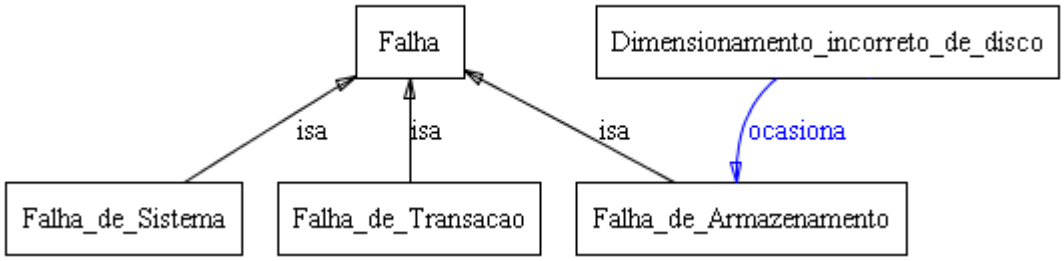

Figura 4-6: Classe Falha, suas subclasses e seu relacionamento com um problema

Dicionario

É a classe que corresponde aos dicionários de mensagens e de parâmetros do SGBD. A classe **Dicionario** armazena as mensagens que identificam os eventos ocorridos no SGBD, quer sejam de aviso ou de erro.

Cada SGBD possui uma estrutura diferente para armazenar essas mensagens [MSSQLServer, 2008], [OraAdmin, 2008], [IBMDB2, 2008], e a criação desta classe permite armazenar mensagens e parâmetros de forma a identificar os eventos indicadores de problemas capazes ocasionar erros ou configurações inadequadas. O principal objetivo dessa classe é fornecer uma estrutura que seja capaz de armazenar tais informações e possuir uma estrutura que mantenha tanto as indicações dos eventos quanto os comandos necessários para a solução destes eventos.

A Figura 4-7 mostra o relacionamento não hierárquico entre as subclasses pertencentes a classe **Dicionário** e a classe **Problema**, uma vez que os problemas encontrados no domínio dos SGBD muitas vezes estão associados às mensagens ou parâmetros desses sistemas.

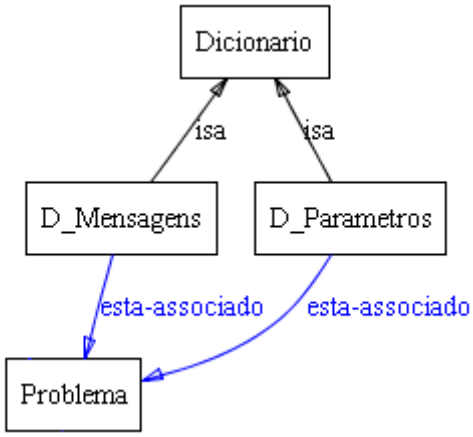

Figura 4-7: Classe **Dicionario,** suas sub-classes e os relacionamentos

Tarefas

Uma tarefa representa um conjunto de instruções e regras que devem ser executadas para impedir a ocorrência de uma falha (proatividade) ou para corrigir seus efeitos (reatividade). Para que essa ontologia pudesse se tornar extensível e reutilizável em SAGBD, as tarefas foram classificadas como autônomas (executadas por agentes de software) e manuais (executadas pelo ABD).

A classe **Tarefas** mantêm relacionamento não-hierárquico com a classe **Acoes,** que serve como maneira de classificar as ações que serão tomadas em virtude dos problemas que necessitam da execução dessas tarefas. Para essa ontologia, admite-se que o cenário ideal é que as os agentes sejam capazes de realizar o máximo de tarefas (englobando ações proativas e reativas), deixando a cargo do ABD apenas as tarefas que ocasionalmente necessitarem de interação humana.

Dessa maneira, conforme indicado na Figura 4-8, é possível identificar o recurso necessário para a execução da tarefa (ABD ou agentes de software) e classificar o tipo de ação necessária a essa ocorrência no futuro, ampliando assim as características de proatividade do SAGBD.

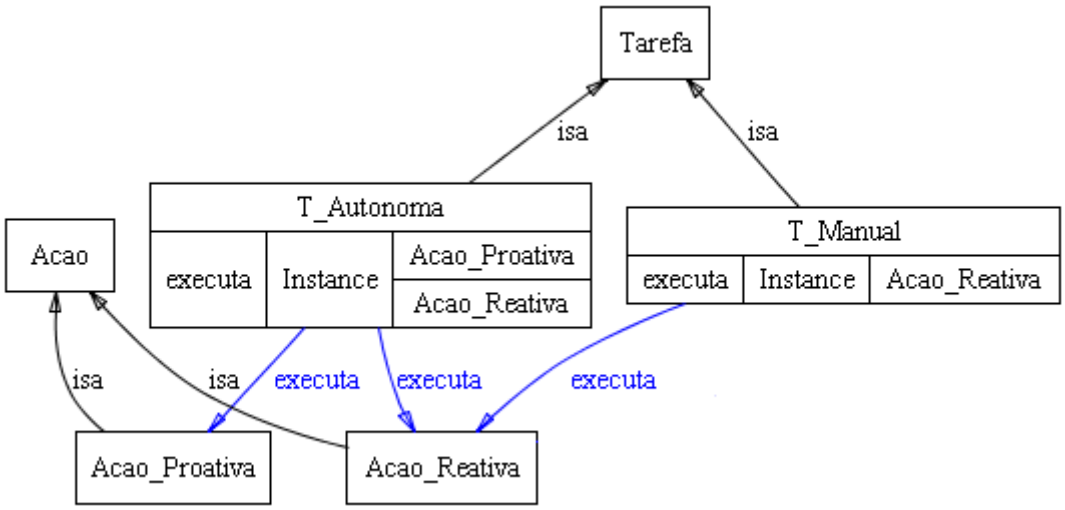

Figura 4-8: Classe **Tarefa**

Acoes

Como demonstrado na classe **Tarefa**, as ações associadas às tarefas podem ser do tipo reativas ou proativas. Essa classe possui três objetivos na ontologia: classificar o tipo de tarefa necessária para resolver determinado problema, categorizar o erro de acordo com a natureza do problema e, por fim, identificar junto com a classe **Tarefa** o recurso necessário para a tarefa solicitada.

A classe **Acao_Proativa** é relacionada à classe **T_Autonoma**, pois o objetivo desta ontologia é apresentar um modelo onde problemas sejam resolvidos com ações proativas antes que se transformem em erros ou falhas. Como a classe **Acao_Reativa** também está relacionada as tarefas autônomas, pode-se concluir que esta ontologia apresenta uma estrutura que beneficia tarefas autônomas tanto reativas quanto proativas.

Nesse contexto, esta ontologia é formalizada de modo que as tarefas manuais sejam executas apenas de forma reativa, deixando o ABD livre da tarefa de prever situações problemáticas que possam ocasionar os erros. A Figura 4-9 apresenta os dois tipos de ações (proativa e reativa) e as correlaciona com as classes as quais elas inferem as soluções. Ou seja, as ações proativas são utilizadas nas situações em que é possível impedir a ocorrência de falhas, portanto, aplicadas aos Problemas. Já as ações reativas são adotadas após a ocorrência da falha e em situações onde ações proativas não puderam ou não tiveram a ação corretiva.

Por entender que falhas são eventos que já ocorreram então apenas ações reativas podem ser executadas para corrigi-los, e os problemas podem ser resolvidos de com ações proativas.

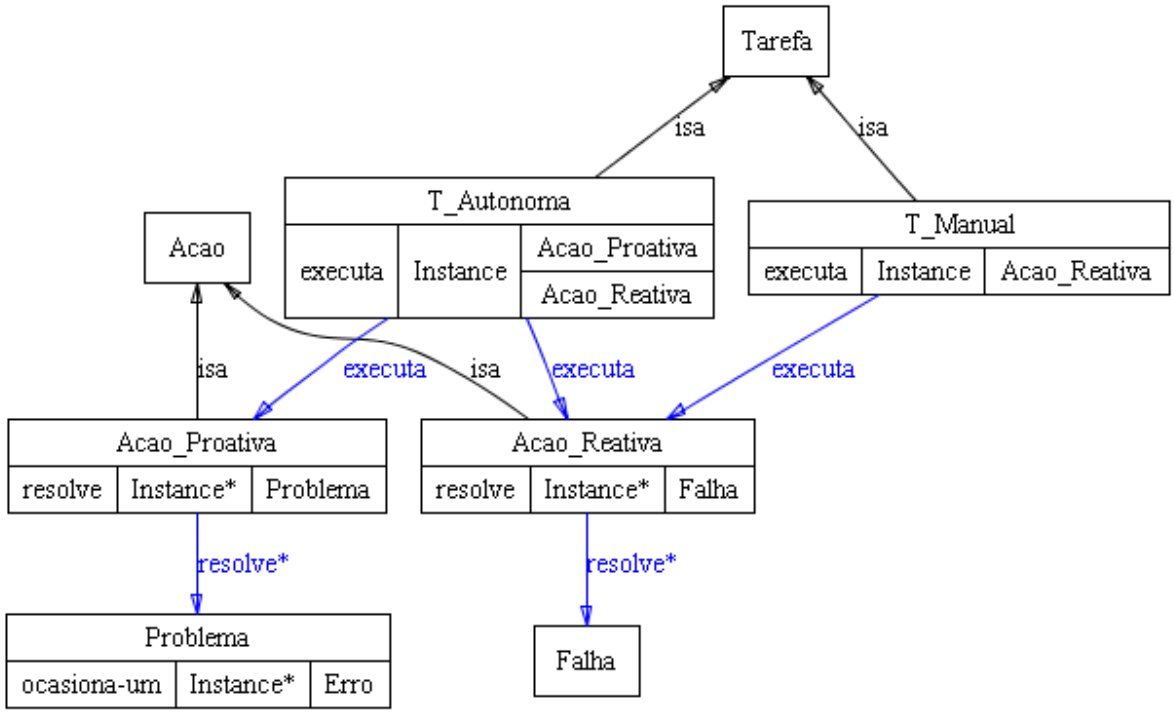

Figura 4-9: Classe **Acao**

4.2.4. Integração de ontologias e classes

Nessa etapa, seguindo a metodologia proposta, ocorre uma busca pelas chamadas meta-ontologias (ou ontologias gerais) selecionando classes que possam ser úteis para garantir que exista uma concordância entre os termos e os conceitos básicos. O próximo passo é localizar bibliotecas de ontologias e verificar similaridades entre as definições. O artefato gerado por essa etapa pode ser uma ontologia oriunda de outras semelhantes, ou então a ontologia originalmente proposta com a adição de classes observadas em outras estruturas ontológicas [Lopez, 1997].

Para a construção da ontologia proposta neste trabalho, não foram localizadas nas bibliotecas de ontologias [SchemaWeb, 2008],[Swoogle, 2008],[OntoSelect, 2008], uma ontologia de domínio correspondente as necessidades de um SAGBD, por isso a ontologia aqui proposta nesse trabalho se baseia nos conceitos da engenharia de software para classificar os termos relacionados a problemas, erros e falhas. Além disso, essa ontologia utiliza dicionários compostos por mensagens e parâmetros dos SGBD.

O capítulo 5 desse trabalho apresenta exemplos de implementação do sistema OntoSitter e o uso que esse sistema faz da ontologia desenvolvida. Para que o comportamento dos agentes de software pudesse ser melhor visualizado, foi necessário utilizar um

gerenciador de agentes chamado JADE [Jade, 2008]. A Figura 4-10 mostra as classes necessárias ao funcionamento de uma ontologia que utilize esse gerenciador.

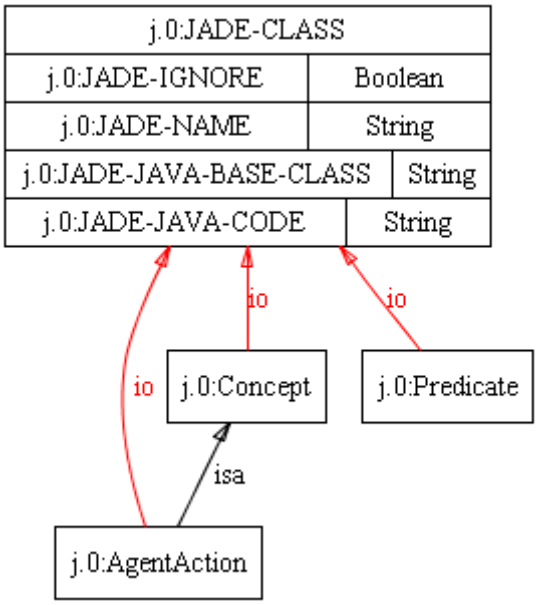

Figura 4-10: Classes necessárias ao gerenciador de agentes JADE [Jade, 2008]

A utilização deste gerenciador exige a integração da ontologia criada com uma estrutura hierárquica de classes que representa o comportamento e identificação desses agentes. A Figura 4-11 apresenta a estrutura de classes necessária a esse gerenciador integrada com a hierarquia de classes da ontologia de domínio criada para esse trabalho.

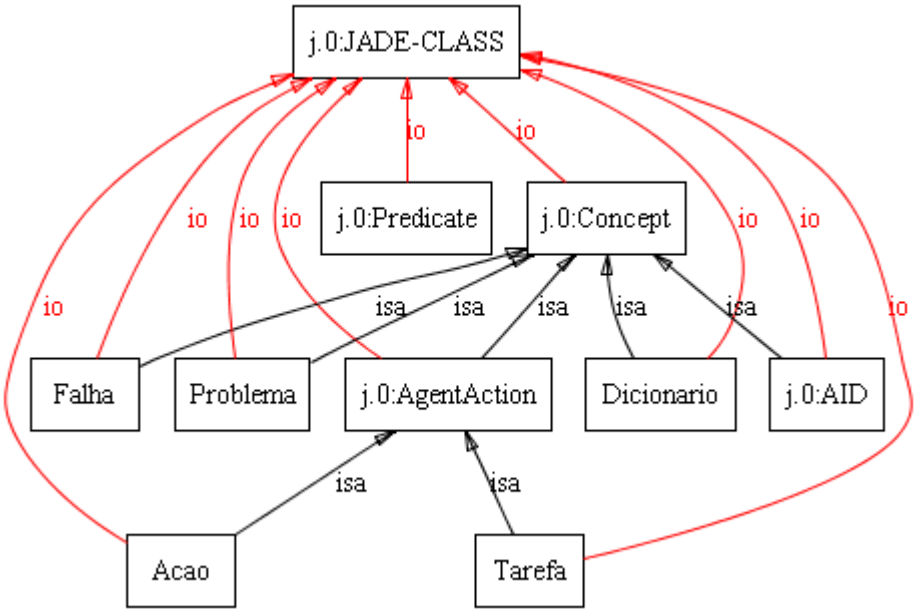

Figura 4-11: Integração de classes do JADE com a ontologia de domínio de SAGBD

Uma ontologia de domínio integrada às classes que representam os agentes inteligentes pode ser exportada pela ferramenta de desenvolvimento de ontologias para classes na linguagem JAVA. O papel dessa conversão consiste em transformar classes criadas em uma ontologia em assinaturas da linguagem de programação, e assim, receber os comandos necessários a criação dos comportamentos e ações desses agentes.

Alguns comportamentos e ações dos agentes constantes nesse trabalho serão apresentados no capítulo 5 desse trabalho. A próxima seção apresenta a etapa de implementação da ontologia originalmente formalizada.

4.2.5. Implementação da ontologia em OWL

A etapa de implementação da ontologia de domínio do OntoSitter apresentada nessa seção define a tradução do modelo conceitual em linguagem OWL-DL. A linguagem utilizada foi definida no Capitulo 3.

A ontologia completa apresenta quatro níveis de profundidade, sendo as classes distribuídas horizontalmente ao longo do primeiro nível, somando seis classes. A classe **Dicionario** mantém um conjunto de indivíduos que possuem características próprias identificáveis através de suas propriedades de anotação. A ferramenta Protégé [Protégé, 2008] disponibiliza três tipos de propriedades que podem ser atribuídas aos conceitos da ontologia criada: propriedades do objeto, propriedades de tipo de dados e propriedades de anotação. Esse último tipo pode ser comparado aos *"slots"* atribuídos as classes da mesma forma que se atribui campos a uma tabela de dados, entretanto, nesse caso as propriedades de anotação são utilizadas para denotar os indivíduos e não as classes.

Somado a esse conceito de relacionamento entre classes, indivíduos e propriedades, ainda aparecem as restrições que complementam a hierarquia da ontologia. Cada um desses itens será a seguir detalhado conceitualmente.

Classes

As classes correspondem a conjuntos de objetos que compartilham as mesmas características do domínio em que estão inseridos, nesse caso o domínio de SAGBD. A Figura 4-12a apresenta as classes **Problema** e **Falha,** representadas através de seus relacionamentos hierárquicos e a Figura 4-12b apresenta as mesmas classes, porém de forma hierárquica e não hierárquica, ou seja, a Figura 4-12a apresenta o relacionamento hierárquico, onde as classes mantém precedência de herança entre elas, o que significa afirmar que as classes são

agrupadas por semelhança ou características próprias. Enquanto isso, a Figura 4-12b apresenta em destaque o relacionamento não hierárquico entre as classes Problema e Falha, que não possuem semelhança direta, porem são relacionados pela propriedade *"ocasiona"*.

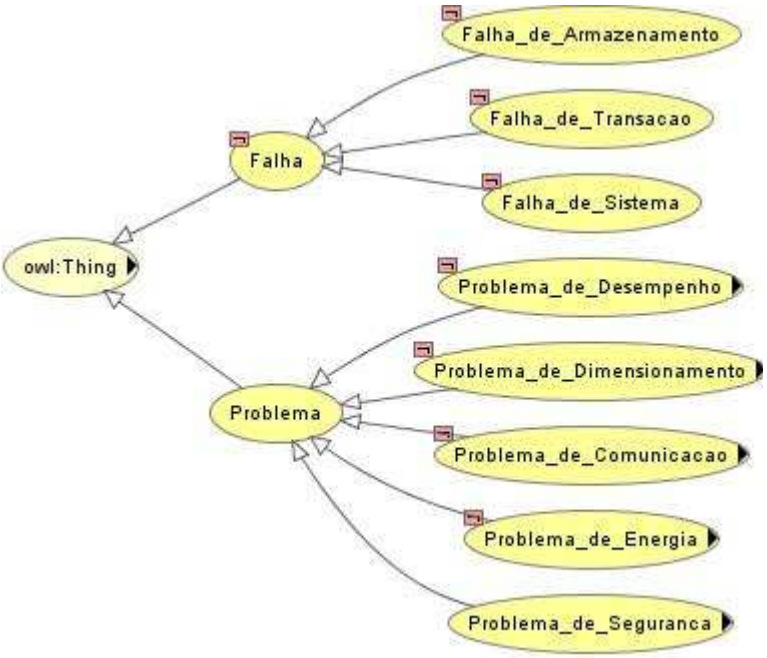

Figura 4-12a: Apenas hierarquia

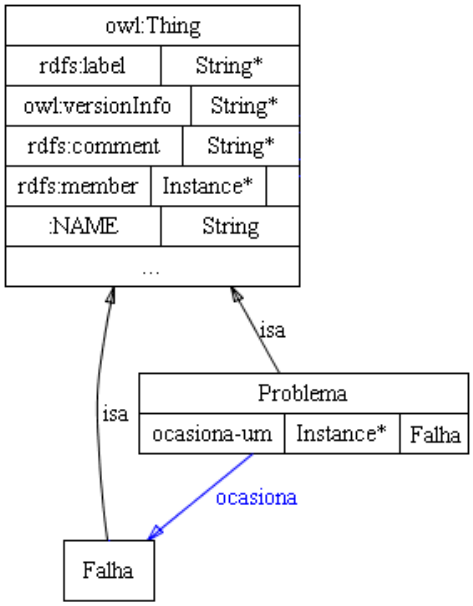

Figura 4-12b: Hierarquia e não hierarquia

Indivíduos

Correspondem aos elementos que pertencem às classes. Neste trabalho os indivíduos representam termos técnicos, mensagens de aviso ou erro e parâmetros dos SGBD. Os *indivíduos* são também conhecidos como *instâncias*. Os indivíduos podem ser referenciados como *Instâncias de Classes*.

Indivíduos representam objetos no domínio de interesse (ou domínio do discurso) e a ferramenta Protégé utiliza um dispositivo chamado UNA-*Unique Name Assumption* (Suposição de nome único) que impede que dois nomes diferentes possam remeter ao mesmo indivíduo.

A ontologia criada nesse trabalho utiliza os indivíduos como forma de armazenamento de conhecimento para representar as mensagens e parâmetros. Na Figura 4-13 são apresentados exemplos de três mensagens de erro de diferentes SGBD e ao lado esquerdo dois exemplos de parâmetros.

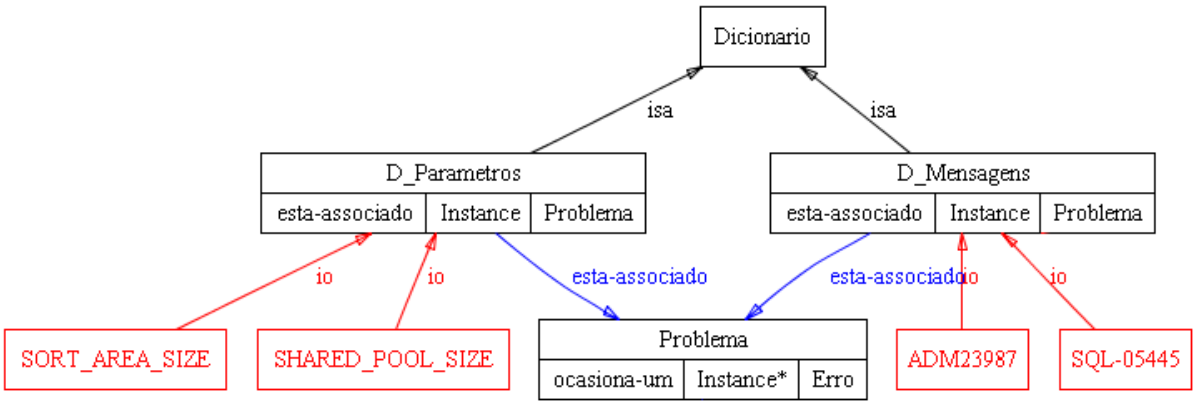

Figura 4-13: Implementação de indivíduos ou instancias de classes

Propriedades

Propriedades são relações binárias entre indivíduos, ou seja, as propriedades servem como elo entre indivíduos ou classes. Por exemplo, a propriedade ***"resolve"*** pode ligar a classe **Acao** com a classe **Problema**. As propriedades podem também ser inversas, como a propriedade **"classifica-como"** é dita propriedade inversa de **"e-classificado-como"**.

As propriedades podem ainda ter um valor único: são as *Functional Properties* (propriedades funcionais), como no caso da propriedade **"possui-proprietario"**. Elas também podem ser Propriedades transitivas ou Simétricas como demonstram respectivamente as propriedades "e-parte-de". A instanciação de propriedades e restrições tem impacto direto nas relações entre as classes e, em especial, nas relações não hierárquicas. A Figura 4-14 apresenta alguns tipos de relacionamentos entre as classes e as propriedades que os originaram: a) Funcional; b) Transitiva.

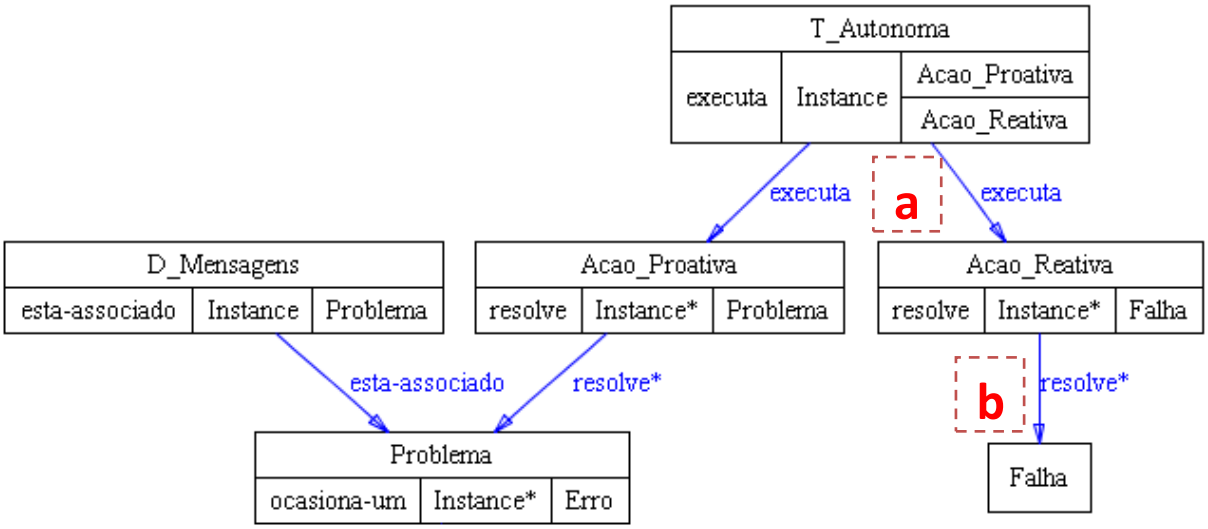

Figura 4-14: Propriedades funcionais e transitivas da ontologia

As propriedades foram utilizadas com três diferentes propósitos nesse trabalho. O primeiro foi o de definir os relacionamentos não hierárquicos entre as classes. O segundo, permitir que fossem criadas restrições baseadas nesse conjunto de propriedades e por fim, fazer com que fosse possível cadastrar informações específicas sobre os indivíduos das classes **D_Mensagens** e **D_Parametros.**

As propriedades e sua utilização junto às restrições serão apresentadas na próxima seção, bem como as figuras que ilustram a utilização dessas propriedades para compor os relacionamentos não hierárquicos.

Restrições

Segundo [Breitman, 2007], em OWL, as propriedades são usadas para criar restrições. Essas restrições são utilizadas para restringir os indivíduos de uma classe. As restrições OWL são classificadas em três categorias: *Quantifier Restrictions* (*Restrições de Quantificador*), *Cardinality Restrictions* (*Restrições de Cardinalidade*) e *Restrições hasValue* (*Restrições temValor*). As *Restrições de Quantificador* são compostas por um quantificador, uma propriedade e uma classe nomeada que contém indivíduos que obedecem a restrição criada. Os dois quantificadores disponíveis são: o *quantificador existencial* em OWL é representado por *someValuesFrom* (*algunsValoresDe*) e o quantificador universal *allValuesFrom* (*todosValoresDe*). A Figura 4-15 apresenta o quantificador universal ***"é-classificado-como"*** e o quantificador existencial ***"esta-associado"***.

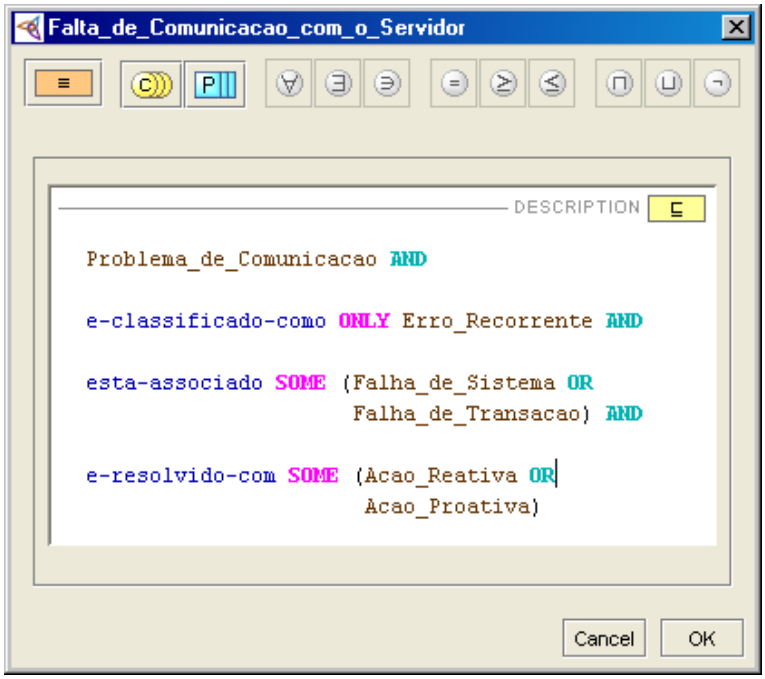

Figura 4-15: Restrições da classe Falta_de_Comunicacao_com_o_Servidor

Na ontologia desenvolvida, a restrição ***Problema_de_Comunicacao ocasiona-um Falha_de_Sistema*** é constituída por um *quantificador universal*, pela propriedade ***ocasiona-um***, e pela classe ***Falha_de_Sistema***. Esta restrição descreve o conjunto, ou a classe que assume que todo problema de comunicação gera apenas a falha do tipo falha de sistema. Esta restrição, bem como as demais restrições são indicadas na Figura 4-16.

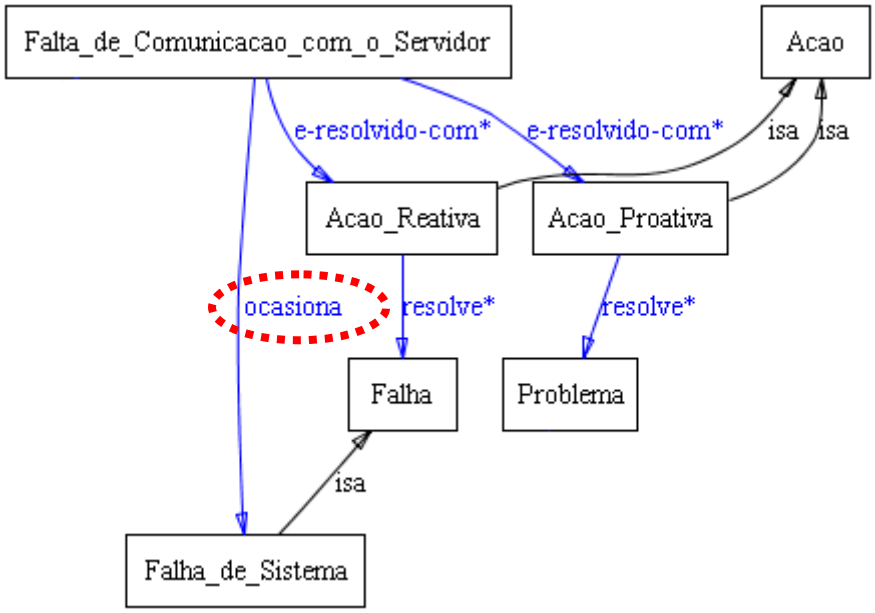

Figura 4-16: Restrições e relacionamentos da ontologia

A próxima seção apresenta as considerações a respeito deste capitulo, sobre a criação da ontologia de domínio e resume as características que permeiam esta ontologia.

4.3. Considerações

O objetivo deste capítulo foi apresentar as etapas de construção de uma ontologia de domínio de problemas para SAGBD. O desenvolvimento do artefato gerado seguiu todas as etapas da Methontology visa ser extensível o suficiente para que novos problemas possam ser facilmente classificados e as ações que impedem sua recorrência sejam identificadas.

Entretanto, quando a detecção de problemas não ocorre, é desejável que um SAGBD classifique esse problema e execute tarefas reativas a fim de solucionar os efeitos causados pela ocorrência da falha, já que sua origem não permite a ação de tarefas proativas pela natureza inesperada que ele possui.

Dessa forma, essa ontologia pode ser utilizada para classificar problemas e direcionar ações, sejam elas proativas ou reativas. Quanto mais proativo um SAGBD puder se tornar,

menor será o numero de tarefas manuais, e conseqüentemente menores a recorrência de erros causados pelo ABD, por isso, a ontologia abrange os aspectos necessários à solução da falha e a detecção de problemas além de direcionar as ações de forma semanticamente clara.

O próximo capítulo apresenta exemplos de utilização desta ontologia por agentes de software.

Capítulo 5
Implementação do OntoSitter

Este capítulo apresenta dois exemplos de implementações do OntoSitter para o SGBD Oracle. O objetivo é demonstrar o uso das características de um sistema autônomo e apresentar os resultados da utilização de uma ontologia de domínio do SGBD Oracle.

Introdução

Este capítulo apresenta a implementação de duas funcionalidades do OntoSitter aplicadas ao SGBD Oracle. Nestas implementações foram apresentados os modelos autônomos que envolvem os quatro conceitos referentes à computação autônoma segundo [Ganek, 2003]: auto-otimização, auto-configuração, auto-recuperação e auto-proteção.

Este capítulo está dividido da seguinte maneira: a Seção 5.1 apresenta a utilização de uma metodologia para o desenvolvimento do sistema OntoSitter, a Seção 5.2 as tecnologias que foram utilizadas pelo ambiente de desenvolvimento e a Seção 5.3 revela os detalhes e resultados da implementação. A Seção 5.4 finaliza o capítulo com as considerações finais.

5.1. Desenvolvimento do sistema OntoSitter

Esta seção detalha a implementação do OntoSitter de acordo com a metodologia SADAAM (*SADAAM – **S**oftware **A**gent **D**evelopment **A**n **A**gile **M**ethodology)* [Clynch, 2007] para definir o sistema multiagentes que fará uso da ontologia de domínio. Os diagramas utilizados pelas etapas de desenvolvimento do OntoSitter são baseados na notação UML (*Unified Modeling Language*) que é padronizado pela OMG (Object Management Group)[1], porém, são estendidos para atender às mudanças de comportamento dos agentes, se assemelhando assim à notação AUML (*Agents Unified Modeling Language*) padronizada pela FIPA.[2]

O OntoSitter foi concebido com base em uma arquitetura modular. Essa arquitetura mantém um ambiente interno composto pela ontologia de domínio, e pelos agentes **aSensor**, **aAtuador** e **aCoordenador** (cuja indicação se dará nesse capítulo em letras grifadas em negrito).

O agente **aCoordenador** é responsável por intermediar a comunicação entre o **ABD** e os agentes. Cabe ainda ao agente **aCoordenador** realizar as buscas na ontologia de domínio. Fica a cargo dos agentes **aSensor** monitorar o SGBD, enquanto os agentes **aAtuador** aguardam por tarefas designadas pelo **aCoordenador**. O **ABD** executa suas rotinas manuais

[1] A notação e sua especificação são encontradas em: http://www.uml.org/

[2] A notação e especificação da AUML é encontrada em: http://www.auml.org/

diretamente no SGBD e pode confirmar tarefas solicitadas pelo agente **aCoordenador** para que sejam executadas pelos agentes **aAtuador**. A Figura 5-1 apresenta essa arquitetura.

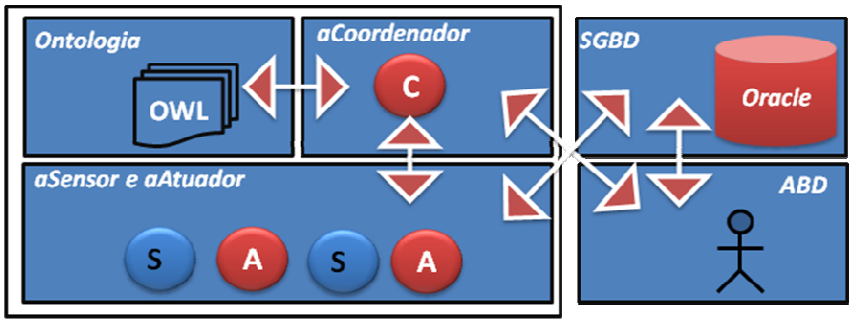

Figura 5-1: Arquitetura do OntoSitter

5.1.1. Projeto

Criação do modelo de comportamento do sistema

Serão apresentadas nessa etapa as tarefas que deram origem ao modelo de comportamento do sistema (do inglês *System Behavioural Model* - SBM) para identificar os comportamentos chave do sistema OntoSitter.

(a) Revisão dos requisitos do sistema e o cenário do negócio: Este cenário possui um agente chamado **aSensor** cuja função é diagnosticar eventos e problemas no SGBD. Esta tarefa é realizada através da leitura constante do arquivo de *log* do SGBD e com a medição de espaço em disco, além de verificar periodicamente o *status* do SGBD, mantendo assim um comportamento cíclico.

Outro agente é chamado de **aCoordenador** que é responsável por receber alertas do agente **aSensor** e buscar na ontologia de domínio de problemas de bancos de dados as informações sobre essa situação para então transformar essa consulta em uma tarefa a ser designada ao agente **aAtuador**. O agente **aAtuador** por sua vez é responsável pela execução de tarefas designadas pelo agente **aCoordenador**, que podem incluir entre outras tarefas a alteração nos parâmetros do banco de dados e a comunicação com o ABD. A fim de representar o comportamento do sistema, foi criado o diagrama de comportamento do sistema e representado pela Figura 5-2.

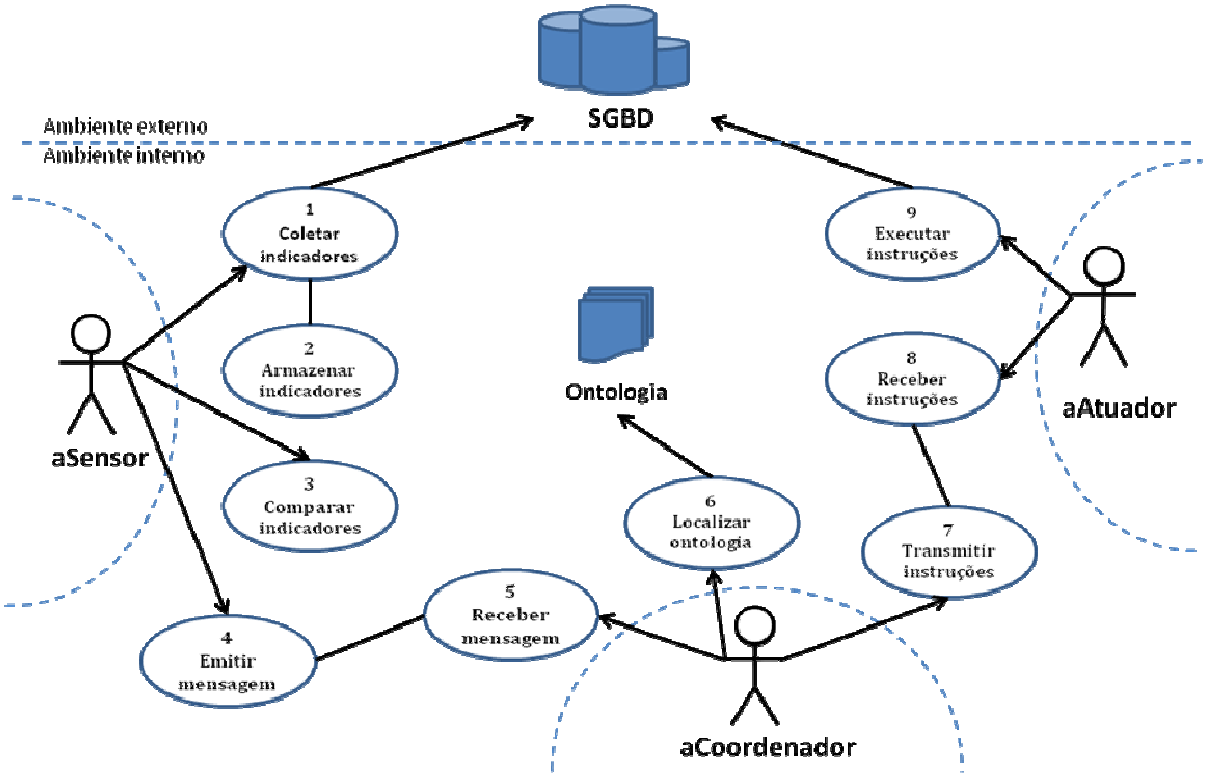

Figura 5-2: Diagrama de comportamento do sistema *OntoSitter*

(b) Definir a estrutura e diagrama organizacional do ambiente: A definição da unidade organizacional para o ambiente OntoSitter foi criada com base no diagrama de comportamento definido (Figura 5-2). Porém, só depois de definir os agentes e as atividades detalhadas deles num sub-nível mais avançado é que foi possível criar os diagramas de unidade organizacional, ou seja, criar sub-níveis que detalhem melhor alguns comportamentos dos agentes apresentados no diagrama de comportamento. A Figura 5-3a e 5-3b apresentam dois exemplos de sub-níveis do diagrama de comportamento do OntoSitter.

Figura 5-3a: Comportamento *leLogEventos*

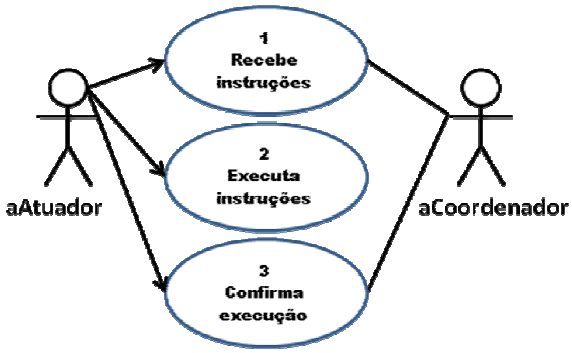

Figura 5-3b: Comportamento *executaAcao*

(c) Atribuir agentes à estrutura organizacional: Nesta etapa foi criada a relação entre os agentes e suas respectivas unidades organizacionais. No diagrama que representa a unidade organizacional do OntoSitter são atribuídos aos agentes seus papéis. A Figura 5-4 apresenta os agentes no contexto do OntoSitter e a relação que eles mantêm com suas atividades.

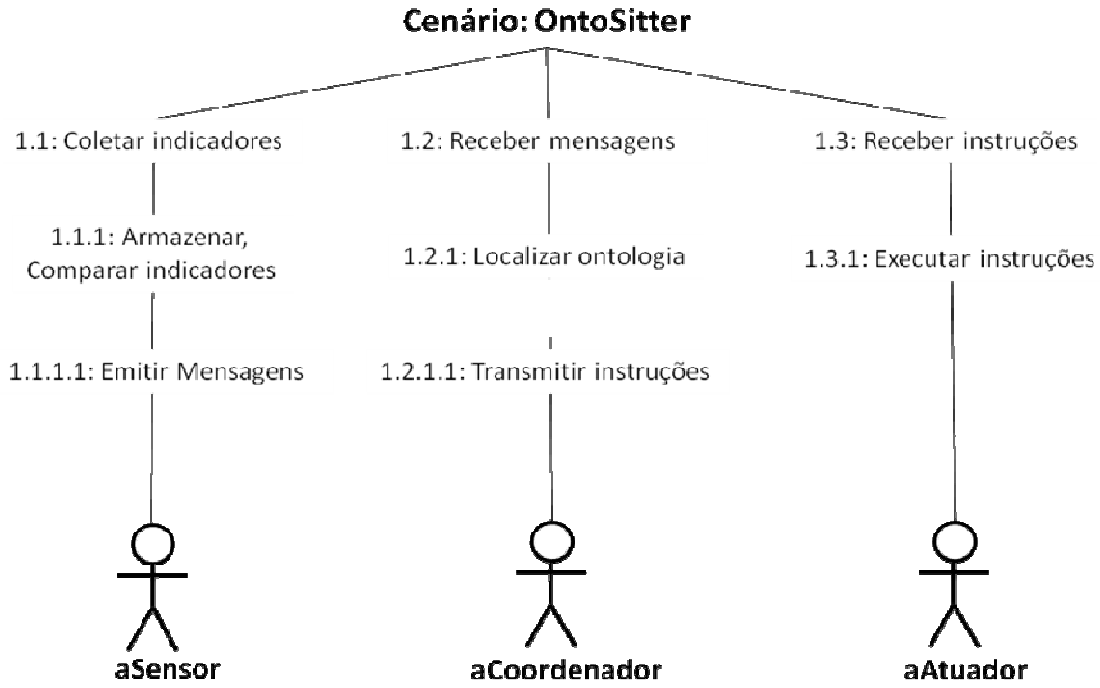

Figura 5-4: Diagrama de unidade organizacional do *OntoSitter*

Selecionar o comportamento do sistema

O comportamento do sistema é determinado pelos diagramas de sub-nível de atividades como indicado na Figura 5-5. Nesse diagrama, os agentes **aSensor** e o agente **aCoordenador** são responsáveis por coletar indicadores do SGBD.

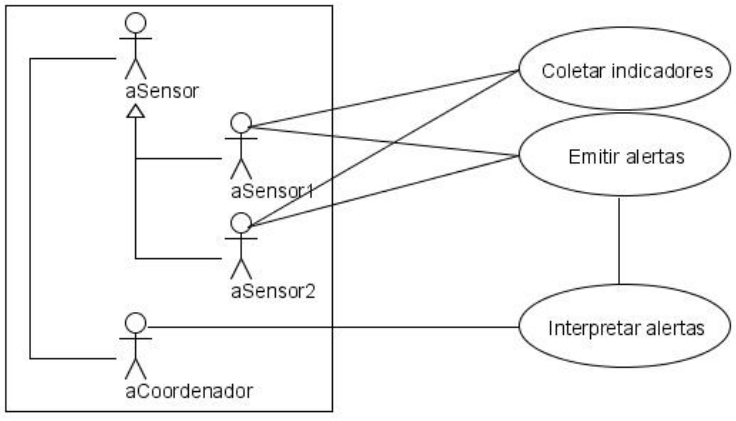

Figura 5-5: Sub-nível do comportamento dos agentes

OntoSitter e DBSitter

Durante essa etapa de determinação dos diagramas de comportamento, foi estabelecida a forma de integração entre o **OntoSitter** e o **DBSitter**, já que ambos compartilham de estruturas semelhantes em relação à hierarquia de classificação de seus agentes de software.

O DBSitter em sua concepção não faz uso de uma ontologia de domínio de problemas, mas sim de um sistema baseado em RBC (Raciocínio baseado em casos) cujo ciclo, segundo [Carneiro, 2004], é composto de quatro etapas: busca, reuso, revisão e retenção (Figura 5-6).

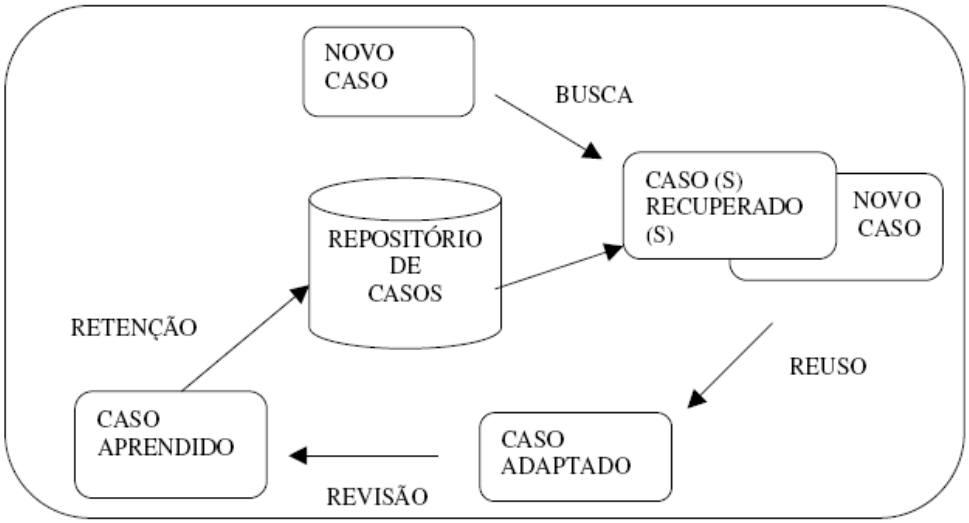

Figura 5-6: Etapas do Sub-nível do comportamento dos agentes [Carneiro, 2004]

Para que o **DBSitter** possa monitorar o ambiente e executar tarefas, ele depende dos Agentes Sensores, Atuadores e Gerenciadores. Segundo [Carneiro, 2004] esses agentes são processos remotos e replicáveis e, portanto, é necessário compor registros que armazenem e identifiquem unicamente cada referência a uma instância específica destes agentes.

Na arquitetura do **DBSitter**, os Agentes Sensores e Atuadores solicitam o registro ao sistema e uma vez que um novo Agente deste tipo é criado no ambiente, ele deve se registrar no sistema para que possa se comunicar e ser identificado.

Ainda segundo [Carneiro, 2004], o Agente Gerenciador de Sensores possui uma área de *cache* onde ficam armazenados os valores da última medição repassada pelos Agentes Sensores. Dessa forma, o sistema não precisa buscar continuamente no Registro de Sensores o endereço de rede do sensor para localizá-lo e coletar sua medição.

O **OntoSitter** foi concebido nos mesmos moldes do **DBSitter,** e utiliza agentes sensores nomeados **aSensor,** agentes que executam tarefas chamados de **aAtuador** e somente

um agente central nomeado **aCoordenador** que é responsável por receber as medições dos agentes **aSensor**, comparar essas medições com os dados constantes na ontologia e designar tarefas reativas ou proativas aos agentes **aAtuador.**

Nesse caso, para que o **DBSitter** passe a utilizar uma ontologia de domínio de problemas em SAGBD, basta que os comportamentos dos agentes criados no **OntoSitter** sejam replicados para sua estrutura, uma vez que ambos os sistemas compartilham das mesmas tecnologias para concepção de sistemas multiagentes.Assim, o ganho que o uso de uma ontologia pode oferecer ao comportamento desses agentes é que eles possuam uma classificação semântica de problemas mais ampla e permita ações proativas baseadas nessa classificação. A Figura 5-7 apresenta uma hierarquia de classes que representa a modelagem destes agentes e o relacionamento que eles mantêm com as classes que representam as tarefas autônomas e as ações proativas.

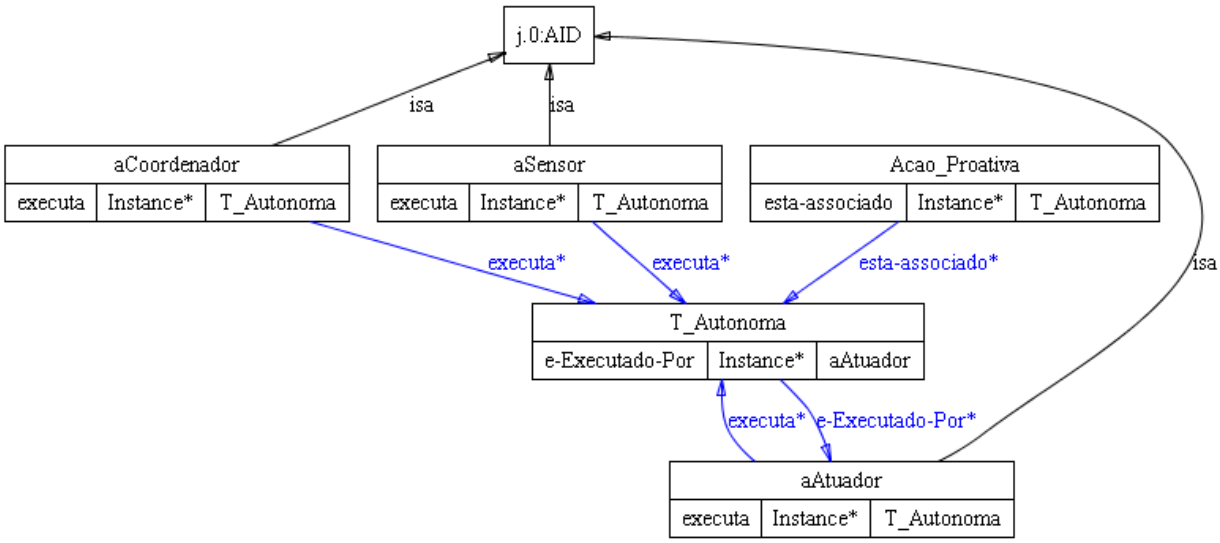

Figura 5-7: Hierarquia de classes e agentes do OntoSitter

Expandir/Definir modelo de interação

A análise do comportamento do sistema mostra que as interações mais freqüentes ocorrem entre os agentes **aSensor** e o agente **aCoordenador**. A Figura 5-8 apresenta um desses cenários de interação onde o agente **aSensor** informa ao agente **aCoordenador** que um evento ocorreu e este, por sua vez, pode não compreender a mensagem, se negar ou então solicitar ao agente **aAtuador** que este efetue determinada ação. A continuação dessa comunicação se dá através de troca de mensagens. O diagrama prevê a possibilidade de vários comportamentos por parte do agente, e esses comportamentos são indicados através da figura composta por um losango grifado com uma letra “X” ao centro.

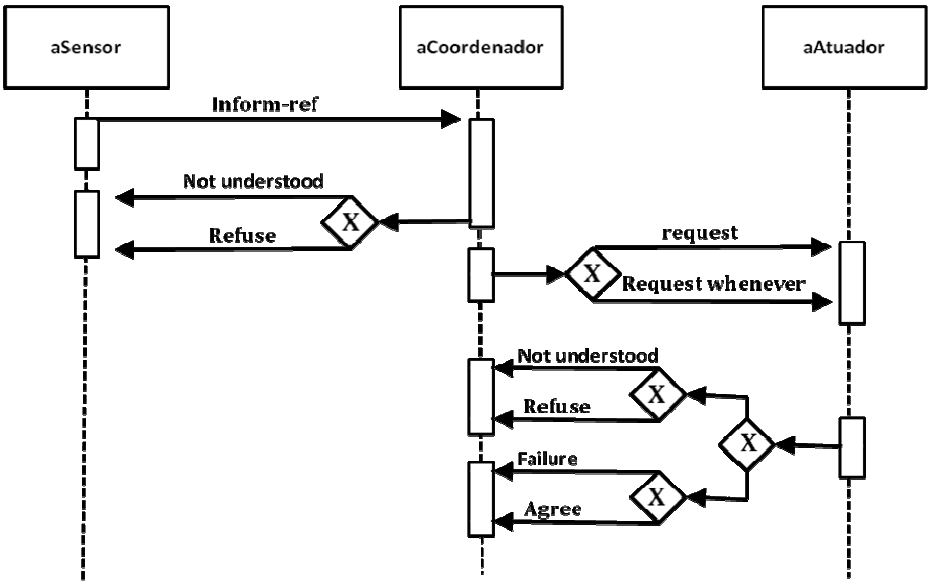

Figura 5-8: Modelo de interação entre agentes do OntoSitter

Expandir o Modelo de Interação (IM) usando o Modelo de Atividades (AM)

No *OntoSitter* foi criado o modelo de atividades com base na principal tarefa do agente **aSensor**: coletar indicadores. Na Figura 5-9 o agente **aSensor** dispara uma mensagem de aviso ao agente **aCoordenador** ao mesmo tempo em que armazena os indicadores coletados no instante em que um evento ocorre no SGBD.

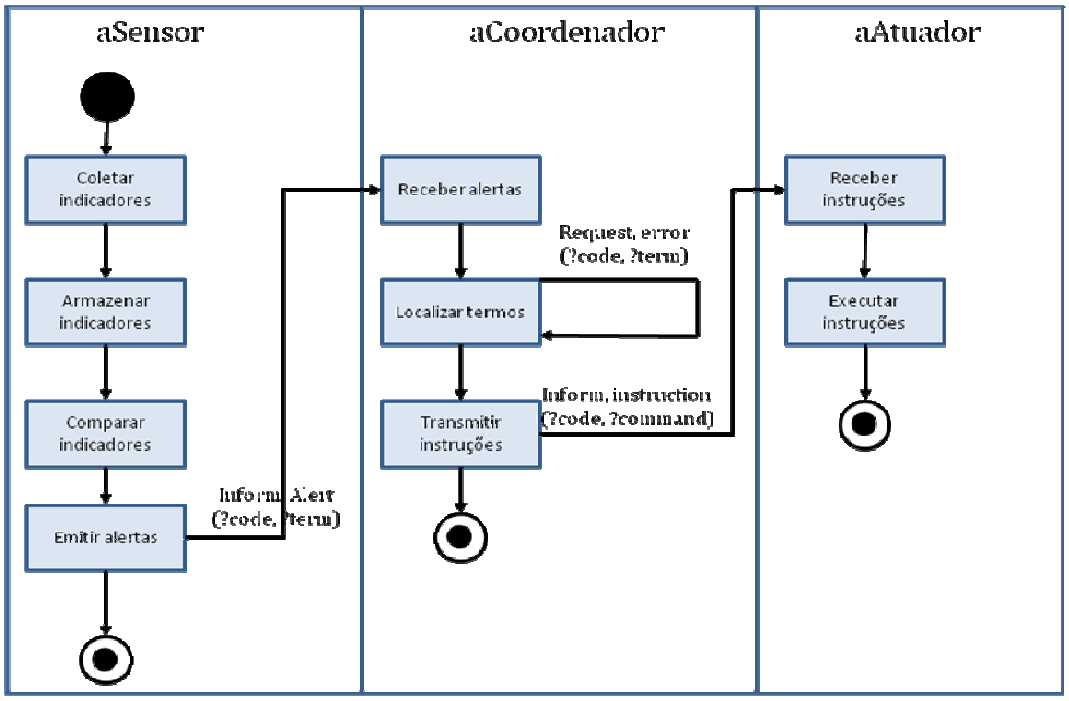

Figura 5-9: Modelo de atividades dos agentes do *OntoSitter*

Após a definição das etapas e dos diagramas da metodologia, o próximo passo é criar a estrutura de agentes de teste que serão responsáveis pelo acesso e recuperação de informações da ontologia desenvolvida.

5.1.2. Implementação orientada a testes

Nessa etapa, para ilustrar a maneira como são executados os testes que atendem a esta metodologia, foram criados o ambiente, os agentes e determinados seus comportamentos. Depois disso, foi criado um agente de teste (TA) com o mesmo nome do agente original (**aSensor**), porém foi acrescido a sua nomenclatura o prefixo "ta", ficando assim grafado: **taaSensor.** Para atender às tarefas dessa etapa foram executados os seguintes passos:

Criação do agente de testes

Como citado anteriormente, foi criado um agente ainda na etapa de desenvolvimento da ontologia de domínio que continha os agentes de teste, pois, quando exportadas as classes da ontologia para classes JAVA, esses agentes de teste também foram exportados com as mesmas características. Optou-se por criar esses agentes por serem mais fáceis de rastrear durante as atividades de seu comportamento cíclico no sistema.

Criar uma aplicação que simula um agente

Após a conversão dos agentes para classes JAVA e adicionados os comportamentos desses agentes, foi feita uma mudança no agente **taaSensor1** para que ele passe a avisar sobre o espaço em disco somente quando houver menos de 1Gigabye de espaço, diferente do comportamento padrão, que já emite avisos ao verificar um percentual menor ou igual a 30% do tamanho total do disco onde está instalada a base de dados do SGBD.

Essa é a função dos agentes AAUT na metodologia SADAAM [Clynch, 2007], através de pequenas mudanças comportamentais, detectar se a execução de tarefas é executada da mesma forma que os agentes que serviram de origem.

Implementar testes e executar

Os agentes foram implementados e testados no ambiente do SGBD Oracle. Os testes com esses agentes diagnosticaram duas situações:

(1) O agente **taaSensor1** executou corretamente seu comportamento e emitiu aviso ao detectar espaço livre em disco inferior a 1Gb e

(2) O agente **taaSensor1** que lê o arquivo de *log* do SGBD não conseguiu permissão de leitura porque o agente **aSensor** já havia executado essa tarefa e não emitiu aviso de erro (Esse agente herdou o comportamento do agente **aSensorRead**, que por sua vez também herdou seus comportamentos do agente **aSensor**).

Através dessa implementação do agente de testes, foi possível corrigir o comportamento do agente **aSensor**, para evitar que a tarefa deixasse de ser executada para esse caso de testes.

Integrar e depurar AAUT

Como citado na tarefa anterior, graças ao teste de agentes foi detectada uma falha no comportamento de um dos agentes (**taaSensor1**) e essa falha serviu para corrigir o código fonte do agente que o originou. A integração tem exatamente esse papel, manter agente de origem e agente cópia no mesmo ambiente para que ambos executem tarefas e apresentem seus comportamentos para análise.

Iniciar um novo ciclo

A cada alteração efetuada o sistema era reiniciado e as mensagens trocadas entre os agentes analisadas no ambiente gráfico do *Sniffer* do framework JADE (*Java Agent Develop Framework*) [TILab, 2003]. A Figura 5-10 apresenta o diagrama de classes que representa o agente **aSensor** e seus agentes de teste **taaSensor1**, **taaSensor2** e **taaSensor3**, todos pertencentes à classe dos agentes e inseridos no mesmo ambiente ainda na etapa de desenvolvimento da ontologia.

Este processo de "clonagem" dos agentes para obter um ambiente de testes mais complexo poderia ter sido criado como mencionado, ainda na etapa de desenvolvimento da ontologia, ou então mais tardiamente dentro do JADE.

Porém essa segunda opção cria agentes idênticos aos originais e o foco da implementação baseada em testes é criar agentes com pequenas modificações a fim de atestar o comportamento desses agentes.

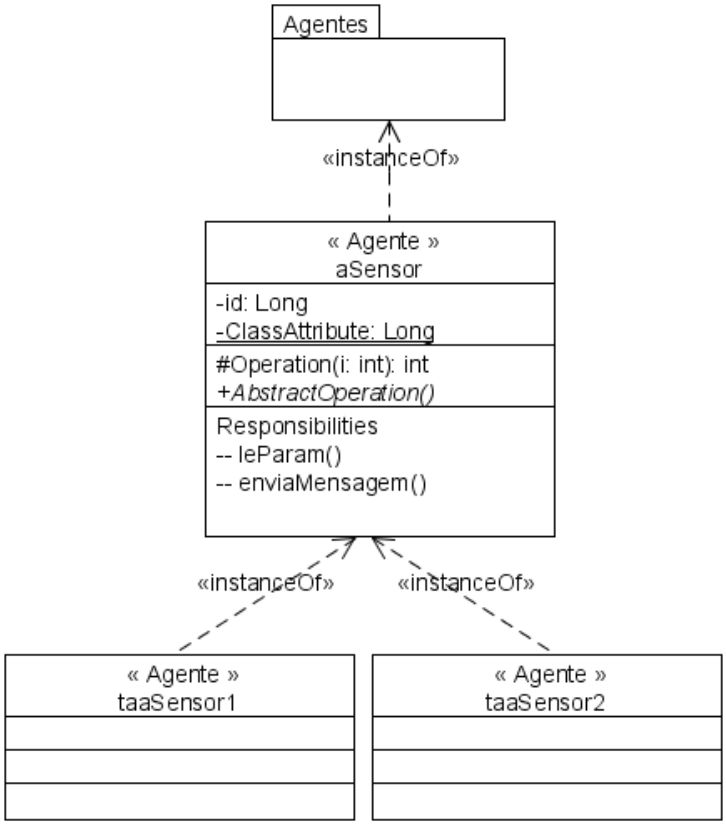

Figura 5-10: Agentes **taaSensor**(n) são instanciados

5.2. Ambiente de desenvolvimento

O Sistema DBSitter, originalmente proposto por Carneiro [2004] utilizou tecnologias de código aberto e apontou como um dos motivos a garantia futura de possibilidade de extensão do produto. Mais tarde Maciel [2007] ao propor seu *framework* também recomendou o uso de tecnologias baseadas em ferramentas de código aberto pela possibilidade de uso em multiplataformas e independência de SGBD.

Esse trabalho, assim como os anteriormente citados, se pauta pelas mesmas práticas e utiliza tecnologias de código aberto por entender que esse tipo de ferramenta colabora com a reutilização e compartilhamento de informação, assim como ocorre com as ontologias.

Para compor o ambiente de desenvolvimento utilizado pelo OntoSitter, foram utilizadas as ferramentas discutidas abaixo.

a) Protégé [Protégé, 2008]: foi o ambiente utilizado para a criação e edição das ontologias. Foi utilizada a versão 3.3.1 por ser a última homologada, já que a versão subseqüente (4.0) não apresentou estabilidade durante testes realizados para a carga de indivíduos e conversão da ontologia para classes Java;

b) Pellet [Pellet, 2008]: é o mecanismo de inferência de raciocínio responsável por checar a consistência da ontologia e verificar a validade das restrições criadas. Foi utilizada a versão código livre 1.5.2 e apresentou rapidez nas inferências embora consuma elevados recursos de memória.

c) SGBD Oracle [Oracle, 2008]: esta versão especifica, (10g) do SGBD foi escolhida por ser gratuita, por sua representatividade comercial, e também pelo interesse nosso interesse em manter o ambiente original proposto para o DBSitter [Carneiro, 2004], facilitando assim a implementação das novas características de uso de ontologias;

d) Eclipse [Eclipse, 2008]: ambiente de desenvolvimento que utiliza a linguagem Java.

e) Java [Java, 2008]: é quase um consenso entre os desenvolvedores e pesquisadores que necessitam de uma linguagem de programação robusta e de código aberto, por esse motivo e por ser a linguagem do ambiente de desenvolvimento citado ela foi escolhida;

f) SGBD MySQL [MySQL, 2008]: o trabalho de Carneiro [Carneiro, 2004] cita o uso do MySQL como banco de dados interno para gravação das atividades de sensoriamento do DBSitter e de armazenamento da estrutura interna do sistema e

g) JADE [Belifimine et al 2003]: é o ambiente que provê a criação e monitoração dos agentes de **s**oftware criados em JAVA, além de controlar o fluxo de troca de mensagens e atividades desses agentes baseado inteiramente no padrão FIPA-ACL [**FIPA, 2008].**

5.3. Implementações

Para padronizar o desenvolvimento das implementações das rotinas do OntoSitter, o processo para sua concepção adotou a metodologia SADAAM [Lynch, 2007], entretanto, de forma abreviada em relação à redução do número de diagramas da etapa de concepção do Projeto.

A etapa de Projeto será representada pela definição de agentes e papéis, seguida pelo diagrama de atividades, a próxima etapa será a criação da hierarquia ontológica de classes utilizada.

A etapa de Implementação orientada a testes será representada pela descrição do funcionamento da rotina multiagentes e da demonstração da interface codificada. Esses processos e o padrão de linguagens utilizadas são detalhados através da Figura 5-11.

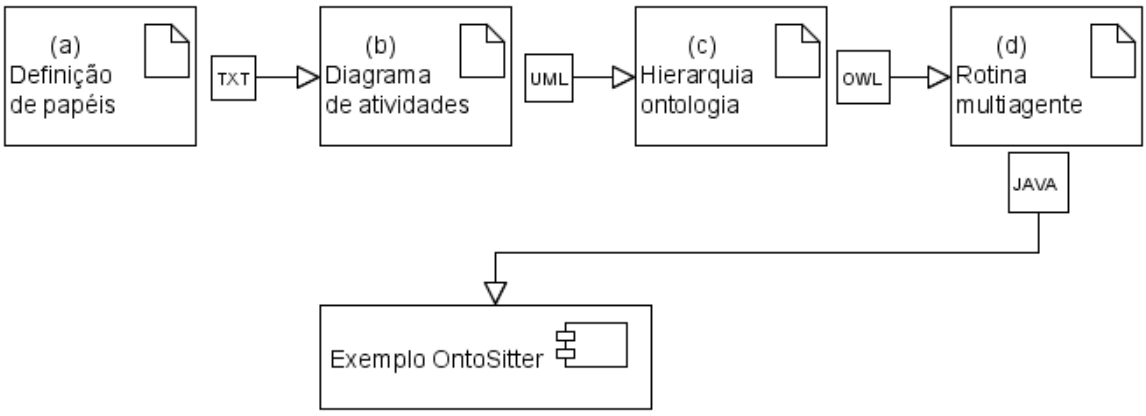

Figura 5-11: Diagrama de processos para criação das implementações

5.3.1. Implementação 1: Interação Agentes- ABD

Esta implementação apresenta uma situação onde o agente **aSensorRead** mantém um comportamento cíclico (***leLogEventos***) avaliando constantemente se foi criado um novo *log* de eventos do SGBD Oracle[3]. A Figura 5-12 apresenta a tela inicial do SGBD, onde consta no canto inferior direito, a informação de que o arquivo de *log* não será armazenado, ou seja,

[3] O SGBD Oracle foi configurado para gerar um novo arquivo de *log* de eventos, sobrepondo o *log* anterior caso ele existisse. Na instalação padrão do SGBD Oracle, este arquivo fica localizado no diretório: C:\oraclexe\app\oracle\admin\XE\bdump\alert_xe.log

será sobreposto a cada novo evento gerado, e o agente **aSensorRead** o apaga após confirmação da execução das tarefas pelo agente **aAtuador**.

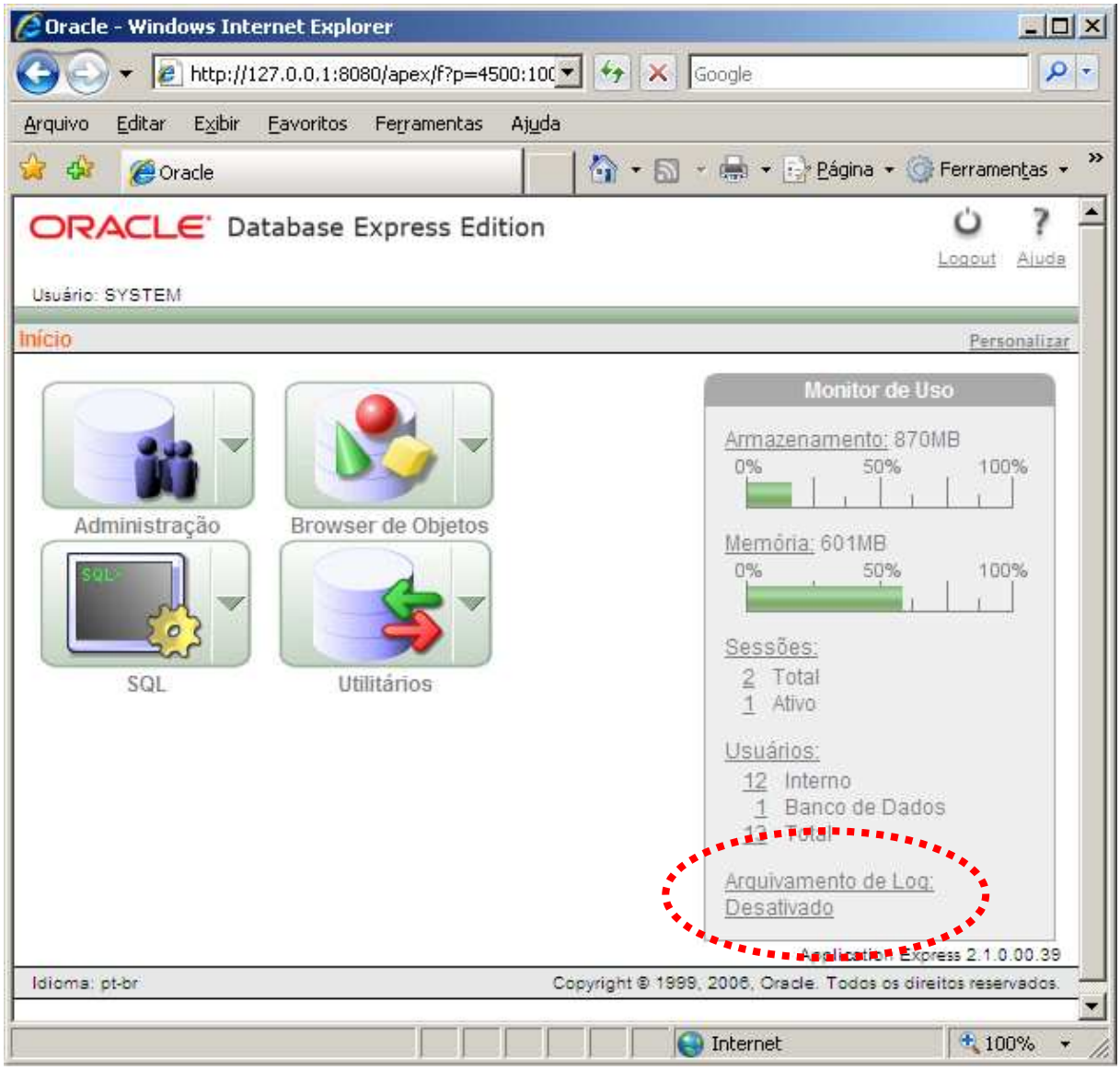

Figura 5-12: SGBD Oracle e o arquivamento de *log* desativado.

Quando é criado um novo arquivo de *log*, o agente **aSensorRead** lê o código e a descrição do evento apontado nesse arquivo e o envia ao agente **aCoordenador** através de uma mensagem FIPA-ACL [FIPA, 2008] .

O arquivo gerado pelo SGBD possui um padrão específico, gravando o código e a descrição do evento na última linha deste arquivo que possui o formato texto, como indicado no Quadro 5-1.

Quadro 5-1: Exemplo de Evento gerado pelo SGBD

```
01 Tue Jun 01 23:50:23 2008
02 RESETLOGS after incomplete recovery UNTIL CHANGE 213418
03 Tue Aug 05 11:27:23 2008
04 Errors in file c:\oraclexe\app\oracle\admin\xe\udump\xe_ora_4968.trc:
05 ORA-00313: abertura falhou para os membros do grupo 1 do thread 1
```

O agente **aCoordenador**, procura no arquivo da ontologia gerado em OWL, pelo código da mensagem e quando encontra o mesmo código (Eg: "ORA-00313"), o agente **aCoordenador** envia uma mensagem ao agente **aAtuador** com o comando SQL contido na ontologia correspondente a esse código e solicita sua execução.

O agente **aAtuador** estabelece conexão com o SGBD através da classe JAVA nomeada ***"oraConnect".*** A implementação desta funcionalidade apresenta o uso de duas características da computação autônoma: a Auto-Configuração e a Auto-Otimização, ambas apontadas por [Kephart, 2003].

Nesse tipo de execução de tarefas os agentes são capazes de identificar a mensagem gerada pelo evento, executar uma consulta na ontologia e solucionar um problema sem a necessidade de intervenção humana. Entretanto, algumas ações podem necessitar do aval do ABD para serem executadas e, esse trabalho apresenta a comunicação entre agentes e ABD.

A ontologia modelada permite definir quais parâmetros e mensagens podem ser executados sem a intervenção do ABD, facilitando assim o processo de auto-diagnóstico e auto-correção por parte do agente **aAtuador**. Essa propriedade de permissão de interação do ABD será apresentada no segundo exemplo de implementação.

A seguir serão apresentadas as fases responsáveis pela definição de papéis, hierarquia da ontologia e descrição do comportamento dos agentes.

a) Definição dos papéis

Segundo Clynch [Clynch, 2007] é nesta fase que utilizando a metodologia SADAAM são definidos os papéis executados pelos agentes. Na metodologia, essa fase pertence à etapa de Projeto do sistema e aqui será apresentado através do Quadro 5-2.

Quadro 5-2: Instâncias e a definição de papéis da implementação 1

Instância	Tipo	Papel e responsabilidade
aSensorRead	Agente do tipo **aSensor**	Ler arquivo de log do Oracle e transmitir conteúdo para o agente **aCoordenador**.
aCoordenador	Agente	Receber mensagens dos agentes **aSensor**; efetuar consultas à ontologia; transmitir mensagens aos agentes **aAtuador** e ao **ABD**.
aAtuador	Agente	Receber mensagens contendo tarefas; executar tarefas; comunicar sucessos ou fracassos ao agente **aCoordenador**.
ABD	Humano	Administrar o SGBD; executar tarefas que o agente **aAtuador** não pode executar.

b) Diagrama de atividades

O Diagrama de atividades detalha o comportamento dos agentes apresentando suas ações durante a execução de algumas tarefas.

A Figura 5-13 apresenta a interação entre os agentes de software e o ABD, além de detalhar as ações para a execução da tarefa de ler o arquivo de *log*, extrair um conjunto de caracteres e compará-lo com os caracteres armazenados na ontologia.

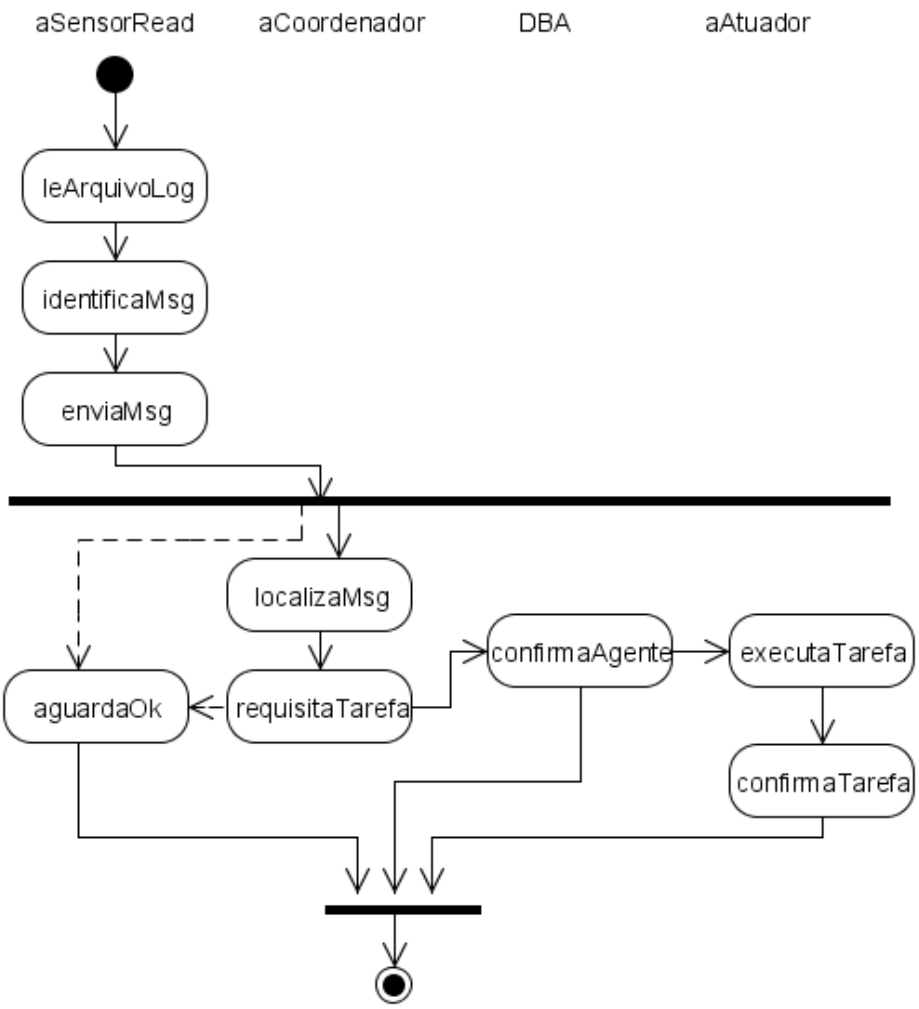

Figura 5-13: Diagrama de atividades da implementação 1

c) Hierarquia da Ontologia

Para atender a esse exemplo de implementação, foram criados dois agentes (**aSensorRead** e **taaSensorRead**) em meio à hierarquia de classes já existente na ontologia e exportadas em classes JAVA, onde foram herdados seus comportamentos dos agentes de origem (**aSensor**).

O framework JADE [Belifimine et al 2003] utiliza um padrão para a definição de ontologias, que determina que a criação de classes obedeça à hierarquia que identifica os comportamentos dos agentes *(j.0:AgentAction)*, os predicados *(j.0:Predicate)* e a identificação *(j.0:AID)*. A Figura 5-14 apresenta a estrutura hierárquica onde as classes necessárias ao JADE são mostradas em destaque.

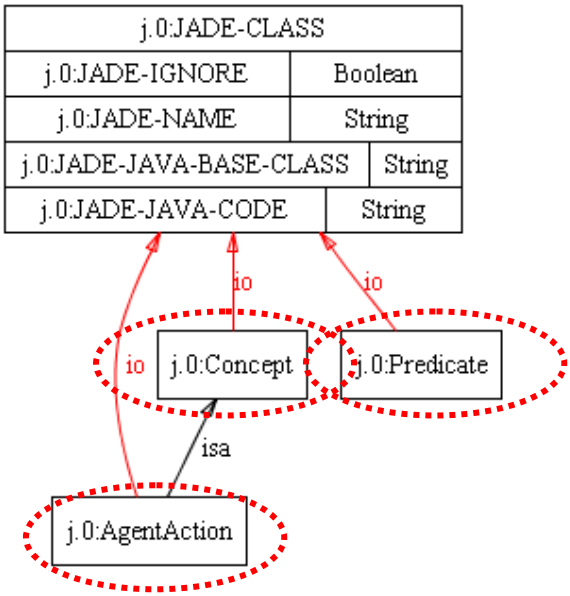

Figura 5-14: Classes necessárias ao framework JADE [Belifimine et al 2003]

Esses agentes foram criados herdando os comportamentos e atributos dos agentes sensores apenas para fim didático, ou seja, para demonstrar o comportamento desses agentes de forma mais clara, sem que tivessem todos eles os mesmos nomes.

Além disso, foram criados indivíduos dentro das classes dos SGBD que identificam o caminho onde o arquivo de *log* está armazenado. Uma restrição indica que existe apenas um arquivo de *log* por SGBD e revela qual o diretório (pasta) onde este arquivo está armazenado, assim o agente **aSensorRead** se torna capaz de ler o arquivo dos SGBD instanciados nessa ontologia.

A Figura 5-14 mostra a tela do Protégé identificando através de quatro janelas as propriedades da classe Oracle.

Para que o agente **aSensorRead** possa ler o arquivo de *log* do SGBD, foi criada uma restrição, com o caminho onde encontrar esse arquivo. Dessa forma, a ontologia pode ser aplicada a quaisquer SGBD que sejam incluídos nessa hierarquia de classes. A Figura 5-15 apresenta a propriedade *caminhoLog* atribuída ao indivíduo **diretório_oracle.**

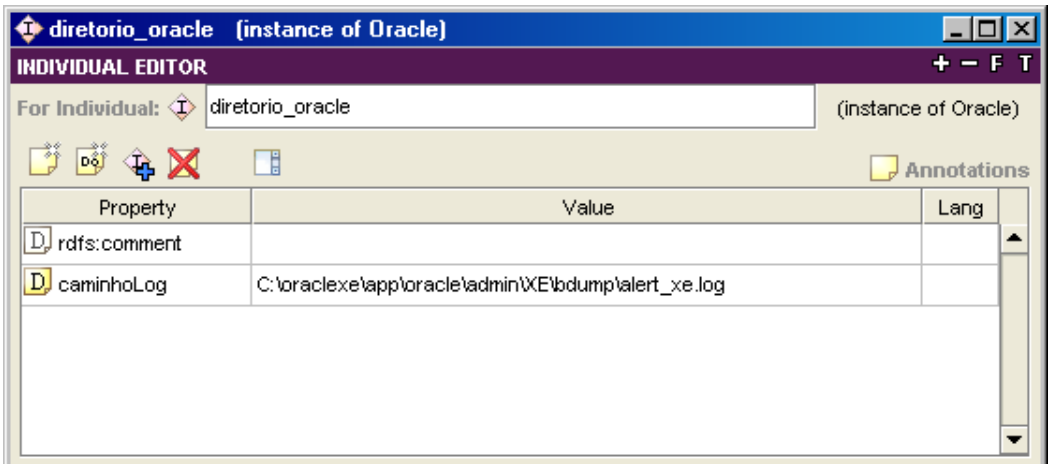

Figura 5-15: Caminho onde o *log* está gravando os eventos do SGBD Oracle

d) Descrição do comportamento dos agentes

O SGBD Oracle foi instalado em um servidor de rede e teve seu parâmetro MAX_SESSIONS alterado para restringir os acessos propositalmente para 5 usuários.

Depois disso foram realizados acessos ao SGBD, até que alguns acessos não foram mais permitidos e os últimos usuários a se conectar ao SGBD receberam uma mensagem informando que o número máximo de usuários foi ultrapassado.

A cada nova tentativa, o Oracle sobrepõe o arquivo de *log* de eventos, que força o agente **aSensorRead** a ler novamente este arquivo e disparar sua mensagem ao agente **aCoordenador**. Esse comportamento pode ser acompanhado através do recurso Sniffer do JADE [Belifimine et al 2003].

Como mencionado anteriormente, a comunicação entre agentes foi estabelecida utilizando o padrão FIPA-ACL [FIPA, 2008] e para isso foram utilizados os campos *conversation-id* e *protocol* para criar os protocolos de comunicação deste exemplo de implementação de rotina que são detalhados no Quadro 5-3.

Quadro 5-3: Relação de Protocolos

Protocolo	Descrição	Valor do campo Conversation-id
Informar_Evento	Mensagem contendo código e descrição do evento ocorrido	Informa_Evento_SGBD
Requisitar_Tarefa	Mensagem com instruções de execução da tarefa relacionada ao evento	Requisita_Execucao_Tarefa

Para tornar mais claro o comportamento dos agentes e os protocolos executados por eles, foi criado um quadro com o fluxo de comunicação desses agentes para visualizar ainda na etapa de Projeto a eventual necessidade de alterações nos códigos dos agentes. O Quadro 5-4 apresenta esse fluxo, onde os agentes **aSensorRead**, **aCoordenador** e **aAtuador** correspondem respectivamente às siglas: S, C e A.

Quadro 5-4: Fluxo de comunicação

Fluxo	Mensagem (ACL)	Conteúdo
S → C	(inform :sender aSensorRead :name aSensorRead@ontomachine:1099/JADE :addresses (sequence ontomachine:1080/jade) :receiver aCoordenador :name aCoordenador@ontomachine:1099/JADE :addresses (sequence ontomachine:1081/jade) :content “(ORA-00018, maximum sessions exceeded)” :conversation-id Informa_Evento_SGBD :language fipa-sl :ontology ontositter)	Agente sensor envia código e descrição do evento localizado no arquivo de log do Oracle.
C → A	(inform :sender aCoordenador :name aCoordenador@ontomachine:1099/JADE :addresses (sequence ontomachine:1081/jade) :receiver aAtuador :name aAtuador@ontomachine:1099/JADE :addresses (sequence ontomachine:1082/jade) :content “(update v$resource_limit set max_utilization=500)” :conversation-id Informa_Evento_SGBD :language fipa-sl :ontology ontositter)	Agente coordenador envia tarefa a ser executada pelo agente atuador

Durante a execução da rotina, o agente **aSensorRead** leu o arquivo de *log* do Oracle, e transmitiu ao agente **aCoordenador** o código e a descrição do erro corretamente.

Depois disso o agente **aCoordenador** leu o arquivo da ontologia e localizou o individuo "ORA-00018" associado à classe **D_Mensagens.** O comportamento **leOntologia** atribuído ao agente **aCoordenador** estava instanciado para ler o conteúdo da propriedade de notação **possuiComando.**

Esse comportamento foi inicialmente definido ainda na etapa de modelagem da ontologia e mais tarde ganhou representatividade ao ser exportado para uma classe JAVA.

A função dessa classe que representa o comportamento do agente é ler o conteúdo desse campo e transmitir uma mensagem ao agente **aAtuador**, para que ele mantenha conexão com o SGBD e execute o comando SQL contido nesse campo.

O Quadro 5-5 apresenta o indivíduo "ORA-00018", sua instrução de comando em SQL e a descrição do evento associado a este código.

Quadro 5-5: Ontologia OWL contendo o indivíduo ORA-00018

```
01<D_Mensagens rdf:ID="ORA-00018">
02      <possuiCodigo rdf:datatype="&xsd;string">ORA-00018</possuiCodigo>
03      <descreveSolucao rdf:datatype="&xsd;string"
04         >Increase the value of the SESSIONS </descreveSolucao>
05      <possuiComando rdf:datatype="&xsd;string"
06         >update v$resource_limit set max_utilization=500</possuiComando>
07         > maximum sessions exceeded.</descreveEvento>
08</D_Mensagens>
```

Foi desenvolvida uma interface no ambiente Eclipse [Eclipse, 2008] com o objetivo de ilustrar as ações dos agentes baseadas nas classes definidas na ontologia de domínio e para que o ABD possa interagir em algumas situações com os agentes.

Nessa interface são apresentadas as informações sobre o SGBD monitorado, o evento lido pelo agente **aSensorRead** e quando esse evento não pode ser executado pelo agente **aAtuador** (pois foi criado um campo para cada mensagem de erro, que identifica se essa ação pode ou não ser executada diretamente sem a intervenção do ABD).

Essa interface foi desenvolvida para que ABD visualize o evento e a que classes da ontologia esse evento está relacionado. Dessa forma o ABD é capaz de relacionar o evento com a tarefa manual que ele precisa executar. Essa interface é representada através da Figura 5-16.

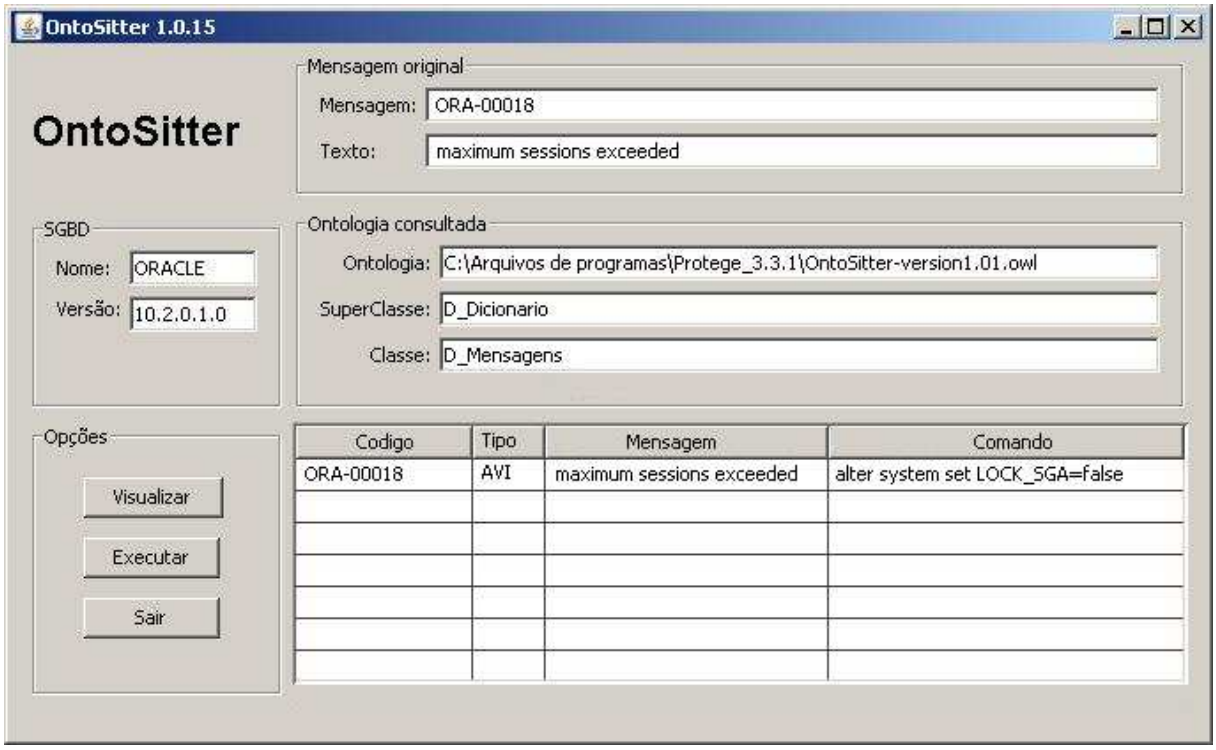

Figura 5-16: Interface do ABD

Assim, esse exemplo de implementação apresenta a utilização da ontologia de domínio por agentes para identificar mensagens do SGBD e realizar uma interação com o ABD para que este decida pela execução ou não dos comandos referentes a essa mensagem.

De acordo com [Ganek, 2003] o que classifica esse exemplo como autônomo, é a capacidade de execução de tarefas baseadas em regras ou políticas, já que cada mensagem pertencente à classe **D_Mensagens** possui o comando necessário a execução da tarefa, servindo assim como uma regra determinada para cada situação.

O próximo exemplo de implementação apresenta a auto-detecção e auto-correção de parâmetros do SGBD, sem a necessidade de interação com o ABD.

5.3.2. Implementação 2: Auto-proteção e auto-recuperação do SGBD

Esta implementação versa sobre o comportamento dos agentes de software, a interação deles com outros agentes e a utilização de uma ontologia de domínio para corrigir um caso clássico na questão do gerenciamento de SGBD: a necessidade de ajustes em parâmetros para obter um melhor gerenciamento de memória do sistema.

Segundo [OraAdm, 2008], um SGBD precisa estar protegido contra falhas decorrentes de área de memória mal dimensionada.Esse exemplo de implementação transfere a responsabilidade do ABD para as atividades autônomas baseadas no conteúdo dos parâmetros armazenados na ontologia. Para isso serão utilizados agentes que verificam os parâmetros da versão Oracle 10g [Oracle, 2008]:

a) *LOCK_SGA:* Esse parâmetro bloqueia a alocação de memória da SGA (*System Global Area*) em memória física quando definido como "*true*". Isso evita excesso de leitura e gravação do *buffer* de memória do servidor do banco de dados em disco.

b) *SGA_MAX_SIZE*: Representa o tamanho máximo da SGA. É correto afirmar [OraAdm, 2008] que a SGA é usada para armazenar as instruções SQL e as definições de dados executados mais recentemente e, cabe a esse parâmetro definir um tamanho máximo para esse espaço de armazenagem.

c) *SGA_TARGET*: Esse parâmetro é responsável pelo gerenciamento automático de memória por parte do banco de dados Oracle. Caso seja informado conteúdo para esse parâmetro outros parâmetros serão substituídos e a alocação de memória RAM será realizada automaticamente pelo SGBD.

As duas principais áreas para medição do tamanho do SGA segundo a [OraAdm, 2008], são o *cache* de dados e *cache* SQL. Configurar esses parâmetros exige um dimensionamento de servidores e cálculos de utilização do SGBD.

Há um consenso informal entre os ABD Oracle de que esses parâmetros para pequenas e médias empresas podem ser evitados com o bloqueio de gravação de arquivos temporários em disco se o parâmetro que bloqueia a alocação de memória da SGA o LOCK_SGA em disco for alterado para "***true"***.

Quando configurado dessa maneira, os *buffers* de memória do servidor do banco de dados utilizam os recursos da memória lógica do servidor e evitam processos desgastantes de escrita e leitura em disco. O Quadro 5-6 apresenta as instâncias e a definição de papéis.

a) Definição de papéis

Diferente do exemplo de implementação anterior, este exemplo não possui um papel definido para o ABD, pois essa funcionalidade identifica e corrige automaticamente os parâmetros necessários ao desempenho do SGBD sem a necessidade de confirmação por parte do ABD

.

Quadro 5-6: Instâncias e a definição de papéis da implementação 2

Instância	Tipo	Papel e responsabilidade
aSensorParam	Agente do tipo aSensor	Ler parâmetros considerados importantes segundo [Oracle, 2008] para melhoria de desempenho e avisar ao agente aCoordenador sobre a necessidade de alteração.
aCoordenador	Agente	Receber mensagens dos agentes aSensorParam; efetuar consultas à ontologia; transmitir mensagens aos agentes aAtuador.
aAtuador	Agente	Receber mensagens contendo tarefas; executar tarefas; comunicar sucessos ou fracassos ao agente aCoordenador.

b) Diagrama de atividades

O diagrama de atividades que demonstra a não necessidade de confirmação da execução desta tarefa pelo ABD, o que torna a rotina totalmente autônoma e, por isso, mais rápida, pois não necessita da interface com o ABD, é demonstrado na Figura 5-17.

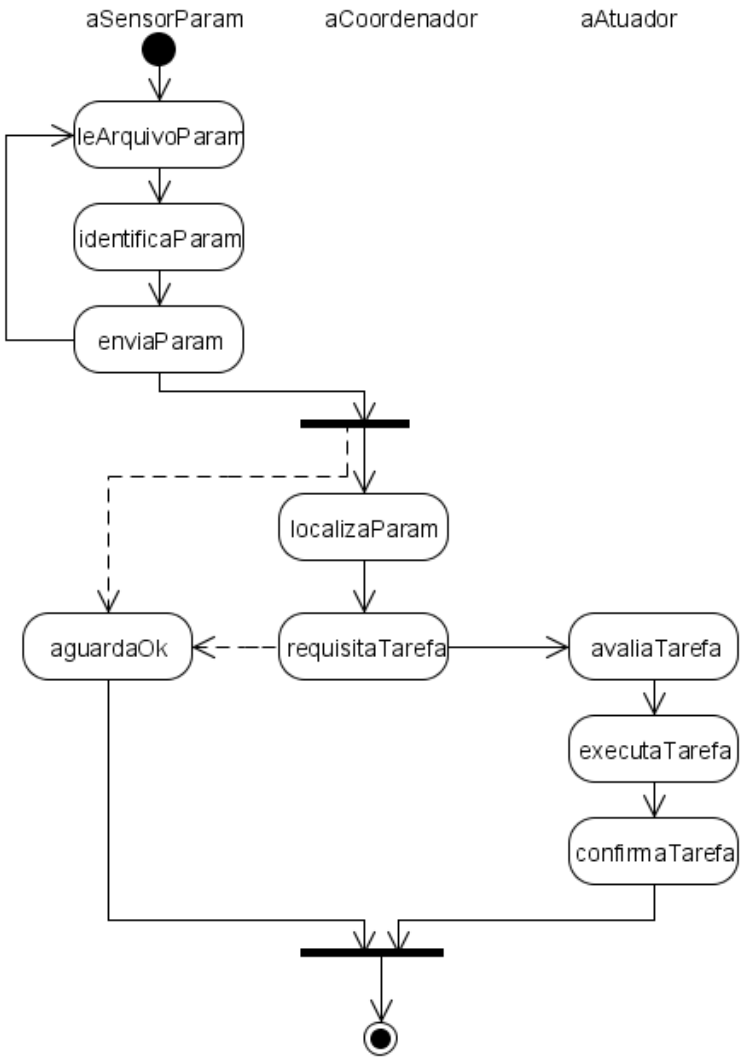

Figura 5-17: Diagrama de atividades da implementação 2

c) Hierarquia da Ontologia

Foram mapeados para a ontologia de domínio os parâmetros mais utilizados para otimização de desempenho do SGBD Oracle [Oracle, 2008]. Aproximadamente 30 parâmetros foram criados indicando o SGBD ao qual pertencem, o nome do parâmetro e o valor ideal através da criação de indivíduos.

A Figura 5-18 apresenta a hierarquia de classes que contém os parâmetros (**D_Parametros**) utilizados por este exemplo de implementação das rotinas do OntoSitter.

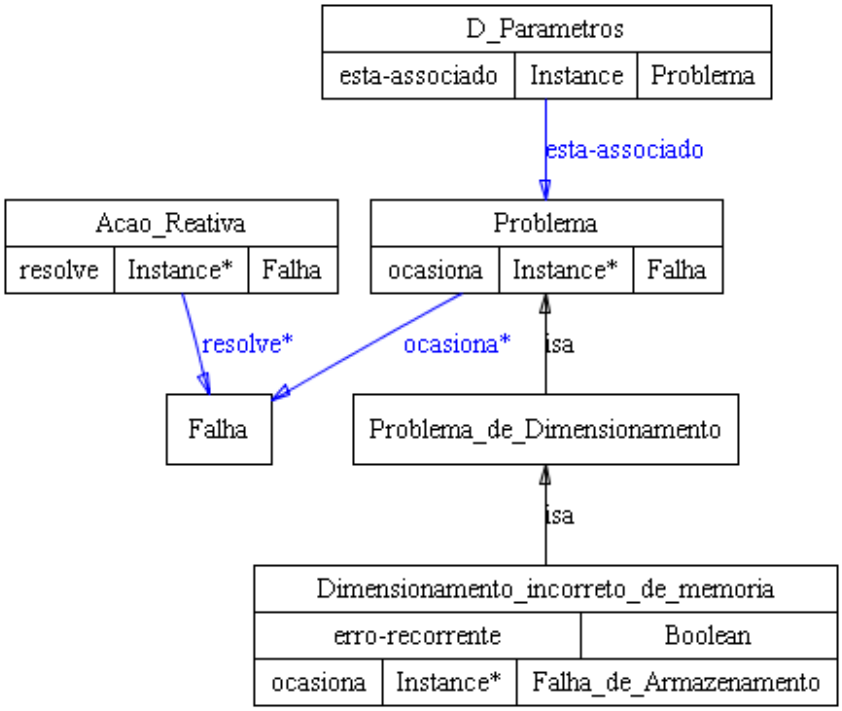

Figura 5-18: Hierarquia de classes para verificação de parâmetros

d) Descrição do comportamento dos agentes

Por padrão o conteúdo do parâmetro **LOCK_SGA** tem seu valor igual a "**false**". Foi deixado o valor padrão e iniciado o ambiente com os agentes. Diferente do estudo de caso anterior, nesse caso não era necessária a interação com o ABD, pois os parâmetros continham o campo **interageABD** com valor igual a ***"false"*.** A Figura 5-19 apresenta a definição do indivíduo instanciado na ontologia para armazenar esse parâmetro. Em destaque na figura, aparece a opção de definir a qual SGBD pertence esse parâmetro, possibilitando estender esse exemplo de implementação a outros SGBD.

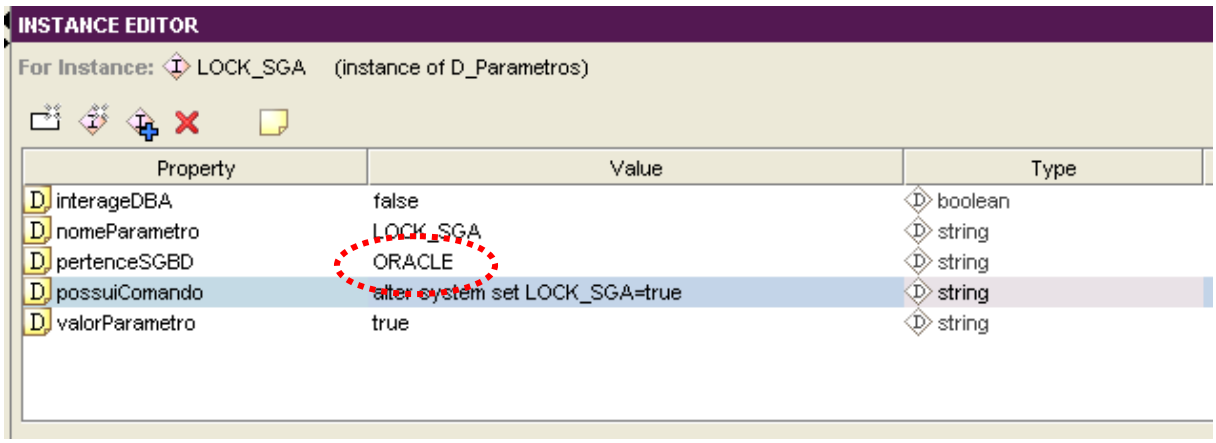

Figura 5-19: Classe **D_Parametros** e o indivíduo **LOCK_SGA**

Com isso o agente **aSensorParam** lê o arquivo de parâmetros do Oracle, detecta o parâmetro citado e altera seu valor. Esse exemplo utiliza a comunicação entre agentes para executar uma tarefa especifica do comportamento atribuído a cada um deles.

Este exemplo de implementação de funcionalidade do OntoSitter apresenta dois conceitos da computação autônoma: a auto-recuperação e a auto-proteção. A auto-recuperação ocorre quando o agente **aAtuador** lê o conteúdo da propriedade **"iniciaSGBD"** e através do comando contido nessa propriedade, o agente é capaz de tornar o serviço do SGBD ativo no estado em que ele estava antes de qualquer evento que por ventura tenha interrompido esse serviço. A Figura 5-20 mostra as propriedades que contêm os comandos responsáveis por iniciar e parar o SGBD Oracle.

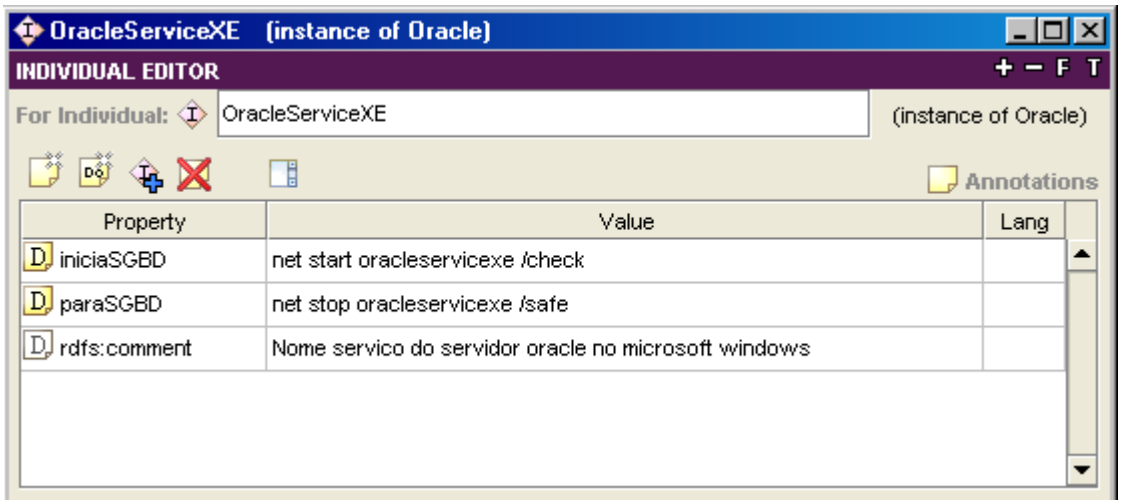

Figura 5-20: Propriedades do Indivíduo "**OracleServiceXE**"

Segundo [Kephart, 2003] a auto-proteção ocorre quando o sistema é capaz de localizar pontos fracos no sistema e corrigi-los antes que se tornem ponto de entrada para falhas. Antes mesmo dos agentes suspenderem os serviços do SGBD o principal comportamento dos agentes é localizar entre os parâmetros quais precisam de ajustes. Esse ajuste protege o SGBD contra falhas de alocação de memória como apresentado nas seções anteriores.

5.4. Considerações

Este capítulo apresentou a metodologia de desenvolvimento de sistemas multiagentes aplicada ao OntoSitter e duas funcionalidades aplicadas a um SGBD.

Esse trabalho se baseia na premissa apresentada por [Ganek, 2003] que define a computação autônoma como "*sistemas que possuem componentes integrados que gerenciam tarefas dinamicamente baseados em regras ou políticas*". Complementando essa premissa, Kephart, 2003, afirma que um sistema é considerado autônomo quando atende aos quatro conceitos básicos: auto-otimização, auto-configuração, auto-recuperação e auto-proteção.

A auto-otimização ocorre nas duas funcionalidades demonstradas, pois os agentes têm a capacidade de alterar parâmetros do SGBD (com ou sem a interação do ABD) para obter um melhor desempenho através de comandos SQL.

A auto-configuração é conseqüência da localização na ontologia dos comandos necessários para a correção de eventos ocorridos no SGBD e sua correta solução.

A auto-recuperação é possível graças ao conteúdo das propriedades e restrições definidas para as subclasses da **D_Parametros** e **D_Mensagens** que determinam quais comandos devem ser executados respeitando o modo de funcionamento e recuperação de dados de cada SGBD.

A auto-proteção é vista no segundo exemplo de implementação, onde os agentes corrigem os parâmetros que indicam o comportamento do sistema frente à alocação de memória. Essa correção autônoma, garante ao sistema funcionamento pleno sem o esgotamento de recursos de memória que podem ser necessários em momentos críticos do SGBD.

Dessa maneira, esse capítulo representou através da implementação de algumas funcionalidades, o uso dos conceitos de computação autônoma e de uma ontologia de domínio de problemas em um SGBD.

Capítulo 6
Conclusões e trabalhos futuros

Este capítulo encerra o trabalho apresentando as considerações finais sobre a utilização de ontologias de domínio de problemas em bancos de dados e indica as perspectivas para que trabalhos futuros possam ser desenvolvidos complementando e estendendo o atual estágio de pesquisa nessa área.

6.2. Contribuições

A contribuição mais valiosa desse trabalho foi criar uma ontologia de domínio que contém uma hierarquia completa de classes relacionadas ao domínio de problemas e erros que ocorrem nos SAGBD. A categorização de problemas pertinentes a esse domínio, é facilmente realizada na medida em que os problemas forem classificados como recorrentes ou inesperados. Outra forma de classificação se refere à forma como as tarefas autônomas ou manuais são designadas para cada tipo de problema.

O nível de classificação e detalhamento das rotinas manuais e autônomas também poderá servir futuramente para compreender melhor as atribuições do ABD em relação às tarefas de gerenciamento de um SGBD, da mesma forma que essa classificação poderá ajudar desenvolvedores na concepção de sistemas multiagentes para a área de bancos de dados.

Utilizar ferramentas de código livre foi uma das premissas desse trabalho, e tem como impacto a contribuição para melhorias nessas ferramentas pela visibilidade que notoriamente elas ganham na medida em que mais trabalhos acadêmicos e comerciais passam a utilizá-las. Dentre as ferramentas utilizadas, a modelagem da ontologia de domínio através do ambiente gráfico fornecido pelo [Protégé, 2008], apresenta vantagens pela padronização de classes e relacionamentos, enquanto a linguagem de programação [Java, 2008] permitiu criar os agentes e seus comportamentos baseados nas classes da ontologia.

A implementação de rotinas autônomas baseadas em classes inicialmente criadas por uma ontologia de domínio, e em seguida transformadas em assinaturas da linguagem de programação só foi possível através da interação que essas ferramentas possuem através de seus *plug-in* utilizados por esse trabalho. Os dois mais utilizados foram o *plug-in Java-Bean-Generator* para a conversão das classes da ontologia em classes Java, e do *plug-in Pellet*, utilizado para inferir raciocínio e checar a definição da ontologia de domínio. Apresentar uma alternativa pautada em ontologias para a aquisição, armazenamento e recuperação de informações também consiste em contribuição ao trabalho desenvolvido anteriormente.

6.3. Limitações e trabalhos futuros

Embora a concepção de um sistema autônomo baseado na arquitetura de dados armazenados em uma ontologia de domínio seja bastante grande, ainda assim, ficam ao longo do trabalho, alguns caminhos que poderiam ter sido melhor explorados e não foram pela questão do tempo e escopo.

Como limitações desse trabalho, podem ser citadas:

- Embora o objetivo não tenha sido substituir a base de conhecimentos pelos indivíduos pertencentes a uma ontologia de domínio, ainda assim, poderia ter sido adicionada uma quantidade maior de indivíduos através de possíveis *plug-in* para o Protégé que não foram encontrados ou se mostraram ineficientes para essa versão em uso;
- O ambiente tecnológico criado para o OntoSitter é composto por um SGBD comercial de grande porte [Oracle, 2008], uma linguagem de programação [Java, 2008] e um ambiente de criação de ontologias [Protégé, 2008]. Esses aplicativos juntos necessitam de recursos de processamento e utilização de memória superior aos tradicionais ambientes de desenvolvimento de sistemas, o que dificultar as tarefas de teste e
- A criação de mais rotinas que atendessem aos conceitos de computação autônoma teria ampliado o espectro de ação do OntoSitter e, com isso, ele poderia deixar de ser considerado um mecanismo de auxílio a SAGBD para também ser classificado como um sistema desse tipo.

Para dar continuidade a essa proposta de utilização de ontologias de domínio pelos SAGBD, podem ser consideradas as pesquisas futuras e melhorias:

- Conversão de classes em indivíduos pelos mecanismos de inferência de raciocínio para atender ao conceito de indivíduos;
- Criação de classes de conexão com outros SGBD além do implementado nesse trabalho;
- Criação de interfaces de interação ABD-Agentes para permitir a visualização das tarefas dos agentes sem a necessidade do JADE e
- Implementação de otimização dos comportamentos cíclicos dos agentes.

6.4. Considerações finais

A recente demanda por sistemas computacionais autônomos tem remetido os desenvolvedores de software à busca por recursos que explorem as características de aquisição de conhecimento, inferência de raciocínio e armazenamento de dados em massa. Se atualmente o ganho que os agentes de software nos propiciam já é uma realidade, é também

real a necessidade de metodologias de desenvolvimento, padrões de projeto e ferramentas de aquisição de conhecimento automatizado.

Este trabalhou procurou explorar a potencialidade oferecida por uma metodologia ágil no ciclo de desenvolvimento de sistemas multiagentes, entretanto, assim como outras metodologias ainda não apresenta a consistência necessária para que o processo seja englobado em sua totalidade. Além disso, o trabalho também apresentou o uso de uma metodologia de desenvolvimento de ontologias que possui o foco e as etapas direcionadas à ontologias de domínio, atendendo assim às necessidades pertinentes desse trabalho.

Desde a definição do escopo inicial deste trabalho, ficou claro que o desafio principal não consistia no uso de tecnologias para desenvolver um produto de software, mas sim na extensão que um mecanismo baseado em ontologias pudesse oferecer aos SAGBD. Tendo o DBSitter [Carneiro, 2004] como exemplo desse tipo de sistema, era necessário apresentar uma alternativa viável para o armazenamento e recuperação de informações sobre problemas recorrentes em SGBD e, para isso, o uso de ontologias se mostrou bastante viável.

Referências

[ACM, 2008] ACM – Association for computing machinery, disponível em http://www.acm.org/ acessado em março de 2008.

[Accioly et al., 2002] Accioly, A. V., Kiyoshi A., Moura, A.M.C.. OntoExtract – Ferramenta para Extração de Ontologias a partir de Bancos de Dados Relacionais. *WTDBD/SBBD/2002*, 2002.

[Agarwal et al., 2003] Agarwal M, Bhat V, Liu H, Matossian V, Putty V, Schmidt C, et al. AutoMate: enabling autonomic applications on the grid. Proceedings of the autonomic computing workshop, fifth international workshop on active middleware services (AMS 2003), Seattle, WA; 2003. p. 22–30.

[Aguiar., 2008] A figura do administrador de bancos de dados. Disponível em: http://www.plugmasters.com.br/sys/materias/313/1/A-figura-do-administrador-do-Banco-de-Dados-(DBA), acessado em março de 2008.

[Aksoyek et al., 1999] Aksoyek S,Aytemir K,Ozer N,Ozcebe O,Oto A, Assessment of autonomic nervous system function in patients with Behcet' s disease by spectral analysis of heart rate variability, Journal of the autonomic nervous system, volume: 77, edição: 2-3 Páginas: 190-4, 24 de setembro de 1999.

[Alani et al., 2003] Alani, H., Kim, S., Millard, D. E., Weal, M. J., Hall, W., Lewis, P. H. e Shadbolt, N. R. (2003) “Automatic Ontology-Based Knowledge Extraction from Web Documents”, em: IEEE Intelligent Systems, vol. 18, ed. 1, p. 14-21.

[Albertin, 1999] ALBERTIN, A. L. Administração de informática: funções e fatores críticos de sucesso. 2. ed. São Paulo : Atlas 1999.

[Almeida, 2003] ALMEIDA, M. B. Roteiro para construção de uma ontologia bibliográfica através de ferramenta automatizada. Perspetivas Em Ci, Belo Horizonte, v. 8, n. 2, p. 164-179, 2003.

[Anabuki, 2008] Anabuki, D. L., Desenvolvimento de uma Plataforma Multi-Agente cognitiva para Bancos de Dados Autônomos: o caso do *DBSitter Plus*. Disponível em: http://www.cin.ufpe.br/~tg/2006-1/dla.pdf . Acessado em março de 2008.

[Araujo, 2003] Araujo, M. (2003) "Educação à distância e a Web Semântica: modelagem ontológica de materiais e objetos de aprendizagem para a plataforma COL". São Paulo. São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.178p. Tese (Doutorado)

[Arellano et al., 2008] Arellano-Garcia, Christian, Storm, Adam, Taylor, Colin, The self-tuning memory manager (STMM): a technical White paper. Disponível em: ftp://ftp.software.ibm.com/software/data/pubs/papers/stmm.pdf, acessado em março de 2008.

[AUML, 2008] Agent Unified modeling language disponível em: http://www.auml.org/ acessado em fevereiro de 2008.

[AutoAdmin, 2008] AutoAdmin: Self-Tuning and Self-Administering Databases.. Disponível em http://www.research.microsoft.com/dmx/autoadmin/ Último acesso em março de 2008.

[Autonomic Computing, 2007] Disponível em http://www.autonomiccomputing.org/ Último acesso em março de 2008.

[Becket, 2003] Dave Beckett. RDF/XML syntax specification (revised). W3C Working Draft, 2003. Available at http://www.w3.org/TR/2003/WD-rdf-syntax-grammar-20030123.

[Blois, 2004] Blois, M.; Lucena, Carlos (2004) "Multi-Agent Systems and the Semantic Web - The SemanticCore Agent-based Abstraction Layer". Proceedings of Sixth International Conference on Enterprise Information Systems ICEIS 2004. Porto: INSTICC, 2004. v.4. p.263 – 270.

[Breitman, 2005] BREITMAN, Karin Koogan. Web semântica: a internet do futuro. Rio de Janeiro: LTC, 2005. 190 p.

[Breuker e Velde, 1994] [Breuker & van de Velde, 1994] J. A. Breuker and W. van de Velde: *CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, The Netherlands.

[Carneiro, 2005] Carneiro A., DBSitter: Um Sistema Inteligente para Gerenciamento de SGBD. 2005. 208p. Dissertação (Mestrado em ciência da computação) – Centro de informática, UFPE, 2005.

[Clynch, 2007] Clynch, N., Collier, R., SADAAM: Software Agent Development - an Agile Methodology, 2007.

[Corcho e Gómez-Pérez, 2000] Corcho, O., Gómez-Pérez, A. A RoadMpa to Ontology Specification Languages. EKAW 2000. Springer-Verlag. 2000.

[Date, 1990] DATE, C. J. Introdução a Sistemas de Bancos de Dados . Rio de Janeiro: Campus, 1990.

[Dean et al., 2003] Dean M, Connolly D, Harmelen F., Hendler J, Horrocks I., McGuinness D., Patel-Schneider P., e Stein, L.A.. Web ontology language (OWL) reference version 1.0. W3C Working Draft, 2003.

[Delamaro, 1996] M. Delamaro, J. Maldonado, and A. P. Mathur, "Integration testing using interface mutations, " Proceedings of the Seventh International Symposium on Software Reliability Engineering, IEEE Computer Society Press, White Plains, New York, pp 112-121, October 30November 2, 1996.

[Felicíssimo et al., 2003] *Carolina Howard Felicíssimo, Lyrene Fernandes da Silva, Karin Breitman, Julio Cesar S. P. Leite.* Geração de Ontologias Subsidiada pela Engenharia de Requisitos. Anais do WER03 - Workshop em Engenharia de Requisitos, Piracicaba-SP, Brasil, Novembro 27-28, 2003, pp 255-269

[Fernandez-Lopez, 1999] FERNÁNDEZ-LÓPEZ et al. Building a chemical ontology using methontology and the ontology design environment. IEEE Intelligent Systems, v. 14, n. 1, p. 37-46, january/february 1999.

[Fipa, 2008] FIPA - Foundation for Intelligent Physical Agents. http://www.fipa.org/

[Firebird, 2008] Conjunto de documentação do Firebird. Disponível em: http://www.firebirdsql.org/manual/pt_br/ Último acesso em março de 2008.

[Freitas, 2002] FREITAS, F.L.G. de. 2002. Sistemas multiagentes cognitivos para a recuperação, classificação e extração integradas de informação da Web. s.l.: UFSC.

[Gabriel, 2000] Gabriel, A., “Metrics involved in the process of evolution”. Disponível em: http://www.fbv.br/gabriel/index_arquivos/frame.htm. Acessado em março de 2008.

[Ganek, 2003] The dawning of the autonomic computing era. IBM system journal, n. 42, p. 5-18, fev. 2003.

[Gava e Menezes., 2003] Gava, T. & Menezes, C. Especificação de Software Baseada em Ontologias. In. III Escola Regional de Informática, 167-205, 2003.

[Gruber, 1993] Gruber, T. A Translation Approach to Portable Ontology Specifications. In *Knowledge Acquisition* 199-220, 1993.

[Gruber, 1995] T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. Presented at the Padua workshop on Formal Ontology, March 1993, later published in International Journal of Human-Computer Studies, Vol. 43, Issues 4-5, November 1995, pp. 907-928.

[Guarino, 1998] Guarino N. Formal Ontology and Information Systems, Proceedings of FOIS'98, Trento, Itália, 6-8 jun. 1998.

[Hariri *et al*., 2006] The Autonomic Computing Paradigm. Cluster Computing. n. 9, ano 3, p. 5-17, Jan. 2006.

[Hayes e Waterman., 1993] Hayes-Roth, F., D.A. Waterman, and D.B. Lenat, *Building Expert Systems*. 1983, Massachusetts: Addison-Wesley.

[Helin , 2002] Heikki Helin, Mikko Laukkanen, Performance Analysis of Software Agent Communication in Slow Wireless Networks, IEEE International Conference on Computer Communications and Networks, 2002, pp 1-8

[Heuser, 2001] Heuser, Carlos Alberto. Projeto de Banco de Dados, Porto Alegre: Instituto de informática da UFRGS, Sagra Luzzato, 2001 Série livros didáticos n.º 4.

[Hewlett Packward., 2008] Disponível em http://welcome.hp.com/country/br/pt/welcome.html Último acesso em março de 2008.

[Horn, 2007] Autonomic Computing: IBM Perspective on The State of Information Technology. IBM Corporation. Disponível em: http://www.research.ibm.com/autonomic. Último acesso em março de 2008.

[Horridge, 2004] HORRIDGE, M. A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools. 2004. Dispon´ıvel em: http://www.co-ode.org/. Acesso em: Março de 2008.

[Hoshino et al., 2007] Online Monitoring of Database Structural Deterioration. In Proceedings of the Fourth international Conference on Autonomic Computing, 2007. Whashington. CD-ROM. Whashington: IEEE, 2007.

[Huhns, 1997] Huhns, M., Singh, M. (1997) Agents onthe Web. IEEE

[IBM, 2008] Disponível em http://www.ibm.com/br Último acesso em março de 2008.

[IBM Autonomic Blueprint, 2006] An architectural blueprint for autonomic computing. Quarta Edição (2006), Disponível em http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf Último acesso em março de 2008.

[IBM DB2, 2008] Disponível em: ftp://ftp.software.ibm.com/ps/products/db2/info/vr95/pdf/pt_BR/db2q0b951.pdf Último acesso em março de 2008.

[IBM Informix, 2008] IBM Informix portfolio: executive overview. Disponível em: ftp://ftp.software.ibm.com/software/data/informix/pubs/whitepapers/informix-roadmap080204.pdf , Último acesso em março de 2008.

[IEEE, 2008] IEEE - IEEE - Institute of Electrical and Electronic Engineers http://www.ieee.org.br

[JADE, 2007] Java Agent DEvelopment Framework. Disponível em: http://jade.cselt.it. Último acesso em março de 2008.

[Jennings, 2001] An agent-based approach for building complex software systems. *Commun. ACM,* 2001.

[Junior, 2008] Junior, Luiz Carlos Ribeiro. OntoLP: construção semi-automática de ontologias a partir de textos da língua portuguesa. 2008. Dissertação (Mestrado em Computação Aplicada) - Universidade do Vale do Rio dos Sinos.

[Kaiser et al., 2003] Kaiser G, Parekh J, Gross P, Valetto G. Kinesthetics eXtreme: na external infrastructure for monitoring distributed legacy systems Proceedings of the autonomic computing workshop, fifth international workshop on active middleware services (AMS 2003), Seattle, WA; 2003. p. 22–30.

[Kaon, 2001] Handschuh, S., Maedche, A., Stojanovic, L., Volz, R., KAON – The Karlsruhe Ontology and Semantic Web Infrastructure, Disponível em: http://kaon.semanticWeb.org/kaon/white-paper.pdf

[Kephart, 2003] The Vision of Autonomic Computing. IEEE Computer Society. n.36, p. 41-50, Jan. 2003.

[Lassila, 1999] Lassila, O.; Swick, R. (eds.). Resource Description Framework (RDF) model and syntax specification.1.0, 22 Feb 1999. (Recomendação do W3C). Disponível em: <http://www.w3c.org/TR/REC-rdf-syntax>. Acesso em março de 2008.

[Lemos, 2002] Alessandra Lemos, Rosario Girardi , Steferson Ferreira, ABARFI, Uma arquitetura reutilizável para a recuperação e filtragem de informação, 2008, online: http://maae.deinf.ufma.br/Artigos/Grupo/ABARFI.PDF

[Lenat, 1990] LENAT, D. B.; GUHA, R. V. *Building large knowledge-based systems*. Massachussets: Addison-Wesley, 1990. 372

[Lifschitz et al., 2004] LIFSCHITZ, S.; MILAN´ES, A. Y. ; SALLES, M. A. V.. Estado da arte em auto-sintonia de sistemas de bancos de dados relacionais. Technical report, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2004.

[Lins, 2007] Lins F. (2007) Modelagem da Mprime Ontology em Lógica Descritiva e Implementação, Trabalho de Graduação - Universidade Federal de Pernambuco. Recife – Brasil.

[Lohman e Lightstone 2002] Lohman, G. M. and Lightstone, S. S. 2002. SMART: making DB2 (more) autonomic. In *Proceedings of the 28th international Conference on Very Large Data Bases* (Hong Kong, China, August 20 - 23, 2002). Very Large Data Bases. VLDB Endowment, 877-879.

[López, 1997] Fernández-López M, Gómez-Pérez A, Juristo N (1997) *METHONTOLOGY: From Ontological Art Towards Ontological Engineering*. Spring Symposium on Ontological Engineering of AAAI. Stanford University, California, pp 33–40

[Macedo, 2000] Macedo, J.A.F., Um estudo de SGBDs baseados em agentes. 2000. 254p. Dissertação (Mestrado em Ciência da computação) - Departamento de informática, PUC-Rio, 2000.

[Maciel, 2007] DBSitter-AS: Um Framework Orientado a Agentes para Construção de Componentes de Gerenciamento Autônomo para SGBD. 2007. 193p. Dissertação (Mestrado em ciência da computação) – Centro de informática, UFPE, 2007.

[Maedche, 2000] A. Maedche and S. Staab, “Discovering Conceptual Relations from Text,” Proc. European Conf. Artificial Intelligence (ECAI-00), IOS Press, Amsterdam, 2000, pp. 321–325.

[Marcones, 1997] MARCONDES, D. *Dicionário oxford de filosofia* . Tradução D. Murcho *et al*. Rio de Janeiro : Jorge Zahar, 1997.

[Martin e Oxman, 1988] Martin J. and Oxman S. "*Building Expert Systems: A Tutorial*", Englewood Cliffs, N.J., Prentice-Hall, 1988

[Martins, 2005] Martins, C. D., Rezende, S. O. Geração Semi-automática de Taxonomias Usando Clustering para Generalização de Regras de Associação. XVIII ENIA, 2005. São Leopoldo – RS

[Martins, 2006] Martins, Camila Delefrate. Construção semi-automática de taxonomias para generalização de regras de associação. 2006. 87p. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC), 2006.

[Menon et al, 2003] Menon, J., Pease, D. A., Rees, R., Duyanovich, L., and Hillsberg, B. 2003. IBM Storage Tank-- A heterogeneous scalable SAN file system. *IBM Syst. J.* 42, 2 (Apr. 2003), 250-267.

[Microsoft, 2008] Disponível em http://www.microsoft.com/brasil/ Último acesso em março de 2008.

[MSSQL Server, 2008] Microsoft SQL Server. Disponível em www.microsoft.com/sql. Último acesso em março de 2008.

[Noy e McGuinness, 2001] Noy, N. F. e McGuinness, D. L. Ontology Development 101: A Guide to Creating Your First Ontology, Knowledge Systems Laboratory, Stanford University, mar. 2001.

[Nwana, 1996] Hyacinth S. Nwana, Software Agents: An Overview, 1996 , The Knowledge Engineering Review, Volume 11, No 3, October/November 1996, pp 205-244

[O´Brien, 2000] O'Brien.P, Zhan C., Domain Ontology Management Environment, Proceedings of 33rd Hawaii International Conference on Systems Sciences, Jan/2000, IEEE Computer Sciences pg.9, Vol. I. 2000

[Océano, 2007] Océano IBM Research IBM Corp. The Océano Project. Disponível em http://www.research.ibm.com/oceanoproject/. Último acesso em março de 2008.

[Oceanstore, 2007] The OceanStore Project., project overview. Berkeley UC. Computer Science division. Disponível em http://oceanstore.cs.berkeley.edu/. Último acesso em março de 2008.

[OMG, 2008] OMG. Unified modeling language (UML). http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.zip, 2008

[Oracle, 2008] Disponível em http://www.oracle.com/global/br/index.html Último acesso em março de 2008.

[Oracle ODM, 2008] ORACLE DATABASE ADMINISTRATOR'S GUIDE 11G RELEASE 1-11.1 – Disponível em: http://download.oracle.com/docs/cd/B28359_01/server.111/b28310.pdf Último acesso em março de 2008.

[Pampa, 2003] Pampa Quispe, Newton Roy Técnicas e ferramentas para a extração inteligente e automática de conhecimento em banco de dados / Newton Roy Pampa Quispe.-- Campinas, SP: [s.n.], 2003.

[Parashar, 2004] Autonomic Computing: Concepts, Infrastructure, and Applications. IEEE Computer Society, Jan. 2004.

[PostgreSQL, 2008] PostgreSQL Documentation. Disponível em: http://www.postgresql.org/docs/ Último acesso em março de 2008.

[Pour, 2006] Expanding the Possibilities for Enterprise Computing: Multi-Agent Autonomic Architectures. In Proceedings of the 10th IEEE on international Enterprise Distributed Object Computing Conference Workshops, 2006. Washington. CD-ROM. Washington: IEEE, 2006

[Progress, 2008] 10.1B: Progress Database administration manual – Disponível em: http://www.psdn.com/library, Último acesso em março de 2008.

[Regoczei e Hirst, 1994] Regoczei, S. and G. Hirst, *Knowledge and Knowledge Acquisition in the Computational Context.* 1994.

[Royce, 1970] Royce, W.W. Managing the development of large software systems: concepts and techniques. Proc. IEEE WESTCON, Los Angeles, CA. (Cap 3), 1970.

[Russel e Norvig, 1995] RUSSELL, Stuart; NORVIG, Peter. Artificial Intelligence: A Modern Approach. Prentice-Hall Inc. New Jersey, 1995

[Russel e Norvig, 2002] Russel, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach – Prentice Hall; 2nd edition. December, 2002.

[Salles, 2004] Salles, M. (2004). .Criação Autônoma de Índices em Bancos de Dados.. Pontifícia Universidade Católica do Rio de Janeiro. Tese de Mestrado em Ciência da Computação.

[Schumacher, 2003] "Security Engineering with Patterns – Origins, Theoretical Model, and New Applications". Lecture Notes in Computer Science (LNCS 2754). p. 29-44.

[Scott, 1991] Scott, A.C., J.E. Clayton, and E.L. Gibson, *A practical guide to knowledge acquisition*. 1991. 509.

[Smith et al, 2004] Smith, M. K; Welty, C; McGuinness, D. L. OWL Web Ontology Language Guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210/, acessado em Abril de 2008.

[Sobral, 2005] Sobral, Josemando M.T.B.; Integrando agentes cognitivos ao DbSitter.Trabalho de Graduação. Centro de Informática – Universidade Federal de PE, 2005.

[Souza, 2006] Souza dos Santos, Eluzaí. Uma Proposta de Integrações de Sistemas Computacionais Utilizando Ontologias / Eluzaí Souza dos Santos. Brasília : UnB, 2006. 106 p.

[Souza, 2008] Souza, Flávio. Caché: banco de dados pós-relacional. Disponível em: http://dpi.ufv.br/downloads/cache/cache.pdf Último acesso em março de 2008.

[Stair, 1998] STAIR, R. Princípios de sistemas de informação: uma abordagem gerencial. Rio de Janeiro: LTC, 1998.

[Sterritt, 2003] SterrittR,BustardDW. Towards an autonomic computing environment. In: Proceedings of IEEE DEXA 2003 Workshops—First International Workshop onAutonomicComputing Systems, Prague, Czech Republic; September 1–5, 2003, IEEE Computer Society Press, p. 694–8.

[Stonebraker, 1976] The design and implementation of INGRES. *ACM Trans. Database Syst.* 1, 3 (Sep. 1976), 189-222. DOI= http://doi.acm.org/10.1145/320473.320476

[Sybase, 2008] Sybase reference manual. Disponível em: http://manuals.sybase.com/onlinebooks Último acesso em março de 2008.

[Taurion, 2003] Taurion, C. Cenários das aplicações móveis para os próximos anos. Mobile Corporate Forum. Curitiba, agosto, 2003.

[Telford, 2008] Telford R., Usability and design considerations for an autonomic relational database management system. Disponível em: http://www.cs.brown.edu/courses/cs227/Papers/IBMSysJournal03.pdf, Último acesso em março de 2008.

[Topol *et al*., 2003] B. Topol *et al*., "Autonomic problem determination: A first step toward self-healing computing systems", IBM White Paper, 2003. Disponível em: http://ibm.com/autonomic/ Último acesso em março de 2008.

[Trumler et al., 2003] Trumler W, Bagci F, Petzold J, Ungerer T. Smart doorplates—toward an autonomic computing system Autonomic computing workshop fifth annual international workshop on active middleware services (AMS'03); June 2003. p. 42.

[Waldrop, 2003] M.M. Waldrop, "Autonomic computing. The technology of self-management", IBM White Paper, 2003. Disponível em: http:/ibm.com/autonomic/ Último acesso em março de 2008.

[Wooldridge, 2002] Wooldridge, M. (2002): Introduction to MultiAgent Systems. Wiley & Sons.

[Yoshihiro, 2002] Yoshihiro Tohma, "Fault Tolerance in Autonomic Computing Environment," *prdc*, p. 3, 2002 Pacific Rim International Symposium on Dependable Computing (PRDC'02), 2002

Anexo A

Anexo A – DMBOK

DAMA-DMBOK Functional Outline

There are 10 functions and 102 activities. Where there are more than 12 activities within a function, activities have been grouped under two or more sub-functions. Three functions have sub-functions, with 8 sub-functions overall (2, 4 and 2).

Functions and sub-functions are named with noun phrases, while activities are named with verb phrases. Each activity is categorized as belonging to one of four Activity Groups:

- **Planning Activities (P)** : Activities that set the strategic and tactical course for other data management activities. Planning activities may be performed on a recurring basis.
- **Control Activities (C) :** Supervisory activities performed on an on-going basis.
- **Development Activities (D) :** Activities undertaken within projects and recognized as recognized as part of the systems development lifecycle (SDLC), creating data deliverables through analysis, design, building, testing and deployment.
- **Operational Activities (O):** Service and support activities performed on an on-going basis.

The outline below is subject to change as each chapter is drafted, but we expect changes to be relatively minor adjustments at this point.

1. **Data Governance**

1.1. **Data Management Planning**

1.1.1. Identify Strategic Enterprise Data Needs (P)

1.1.2. Develop & Maintain the Data Strategy (P)

1.1.3. Establish the Data Management Professional Organizations (P)

1.1.4. Identify & Appoint Data Stewards (P)

1.1.5. Establish Data Governance & Stewardship Organizations (P)

1.1.6. Develop, Review & Approve Data Policies, Standards and Procedures (P)

1.1.7. Review & Approve Data Architecture (P)

1.1.8. Plan and Sponsor Data Management Projects & Services (P)

1.1.9. Estimate Data Asset Value & Associated Data Management Costs (P)

1.2. Data Management Supervision & Control

1.2.1. Supervise the Data Management Professional Staff & Organizations (C)

1.2.2. Coordinate Data Governance Activities (C)

1.2.3. Manage & Resolve Data Related Issues (C)

1.2.4. Monitor & Ensure Regulatory Compliance (C)

1.2.5. Monitor Conformance with Data Policies, Standards and Architecture (C)

1.2.6. Oversee Data Management Projects & Services (C)

1.2.7. Communicate & Promote the Value of Data Assets (C)

2. **Data Architecture Management**

2.1. Develop & Maintain the Enterprise Data Model (P)

2.2. Analyze & Align With Other Business Models (P)

2.3. Define & Maintain the Data Technology Architecture (P) (same as 4.2.2)

2.4. Define & Maintain the Data Integration Architecture (P) (same as 6.2)

2.5. Define & Maintain the DW / BI Architecture (P) (same as 7.2)

2.6. Define & Maintain Enterprise Taxonomies & Namespaces (P) (same as 8.1)

2.7. Define & Maintain the Meta Data Architecture (P) (same as 9.2)

3. Data Development

3.1. Data Modeling, Analysis & Design

3.1.1. Analyze Information Requirements (D)

3.1.2. Develop & Maintain Conceptual Data Models (D)

3.1.3. Develop & Maintain Logical Data Models (D)

3.1.4. Develop & Maintain Physical Data Models (D)

3.2. Detailed Data Design

3.2.1. Design Physical Databases (D)

3.2.2. Design Related Data Structures (D)

3.2.3. Design Information Products (D)

3.2.4. Design Data Access Services (D)

3.3. Data Model & Design Quality Management

3.3.1. Develop Data Modeling & Database Design Standards (P)

3.3.2. Review Data Model & Database Design Quality (C)

3.3.3. Manage Data Model Versioning and Integration (C)

3.4. Data Implementation

3.4.1. Create & Maintain Development & Test Databases (D)

3.4.2. Create & Maintain Test Data (D)

3.4.3. Migrate & Convert Data

3.4.4. Build & Test Information Products ((D)

3.4.5. Build & Test Data Access Services (D)

3.4.6. Build & Test Data Integration Services (D)

3.4.7. Validate Information Requirements (D)

3.4.8. Prepare for Data Deployment (D)

4. Database Operations Management

4.1. Database Support

4.1.1. Implement & Maintain Database Environments (C)

4.1.2. Implement & Control Database Changes (C)

4.1.3. Acquire Externally Sourced Data (O)

4.1.4. Plan for Data Recovery (P)

4.1.5. Backup & Recover Data (O)

4.1.6. Set Database Performance Service Levels (P)

4.1.7. Monitor & Tune Database Performance (O)

4.1.8. Plan for Data Retention (P)

4.1.9. Archive, Retrieve and Purge Data (O)

4.1.10. Manage Specialized Databases (O)

4.2. Data Technology Management

4.2.1. Understand Data Technology Requirements (P)

4.2.2. Define the Data Technology Architecture (P) (same as 2.3)

4.2.3. Evaluate Data Technology (P)

4.2.4. Install & Administer Data Technology (O)

4.2.5. Inventory & Track Data Technology Licenses (C)

4.2.6. Support Data Technology Usage & Issues (O)

5. **Data Security Management**

5.1. Understand Data Privacy, Confidentiality & Security Needs (P)

5.2. Define Data Privacy & Confidentiality Policies & Standards (P)

5.3. Define Password Standards & Procedures (P)

5.4. Design & Implement Data Security Controls (D)

5.5. Manage Users, Passwords & Group Membership (C)

5.6. Manage Data Access Views (C)

5.7. Manage Data Access Permissions (C)

5.8. Monitor User Authentication & Access Behavior (C)

5.9. Classify Information Confidentiality (C)

5.10. Audit Data Security (C)

6. **Reference & Master Data Management**

6.1. Understand Reference & Master Data Integration Needs (P)

6.2. Define the Data Integration Architecture (P) (same as 2.4)

6.3. Implement Reference & Master Data Management Solutions (D)

6.4. Control Code Values & Other Reference Data (C)

6.5. Integrate Master Data (O)

6.6. Replicate Reference and Master Data (O)

6.7. Maintain Dimensional Hierarchies (O)

7. **Data Warehousing & Business Intelligence Management**

7.1. Understand Business Intelligence Data Needs (P)

7.2. Define & Maintain the DW / BI Architecture (P) (same as 2.5)

7.3. Implement Data Warehouses & Data Marts (D)

7.4. Implement Business Intelligence Tools & User Interfaces (D)

7.5. Implement Enterprise Reporting (D)

7.6. Implement Management Dashboards & Scorecards (D)

7.7. Implement Analytic Applications (D)

7.8. Train Business Professionals (O)

7.9. Replicate & Transform Data for Business Intelligence (O)

7.10. Monitor & Tune Data Warehousing Processes (C)

7.11. Support Business Intelligence Activity (O)

7.12. Monitor & Tune BI Activity and Performance (C)

8. **Document & Content Management**

8.1. Plan for Managing Electronic & Physical Documents (P)

8.2. Define & Maintain Enterprise Taxonomies & Namespaces (P) same as 2.6)

8.3. Implement & Maintain Document Storage Systems (D)

8.4. Acquire & Store Documents (O)

8.5. Index Document Information Contents (O)

8.6. Backup & Recover Documents (O)

8.7. Support Document Content Analysis (O)

8.8. Support Document Access, Circulation & Update (O)

8.9. Monitor & Tune Document Retrieval Performance (C)

8.10. Archive, Retrieve & Purge Documents (O)

8.11. Audit Document & Content Management (C)

9. **Meta Data Management**

9.1. Understand Meta Data Requirements (P)

9.2. Define the Meta Data Architecture (P) (same as 2.7)

9.3. Develop & Maintain Meta Data Standards (P)

9.4. Implement a Managed Meta Data Environment (D)

9.5. Create & Maintain Meta Data (O)

9.6. Integrate Meta Data (C)

9.7. Manage Meta Data Repositories (C)

9.8. Distribute & Deliver Meta Data (C)

9.9. Support Meta Data Reporting and Analysis (O)

10. **Data Quality Management**

10.1. Develop and Promote Data Quality Awareness (O)

10.2. Profile, Analyze & Assess Data Quality (D)

10.3. Define Data Quality Requirements & Business Rules (D)

10.4. Test & Validate Data Quality Requirements (D)

10.5. Define Data Quality Metrics & Service Levels (P)

10.6. Measure & Monitor Data Quality (C)

10.7. Manage Data Quality Issues (C)

10.8. Correct Data Quality Defects (O)

10.9. Design & Implement Operational DQM Procedures (D)

10.10. Monitor Operational DQM Procedures & Performance (C)

10.11. Audit Data Quality (C)

Dissertação de Mestrado apresentada por **Ricardo Alexandre Afonso** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"OntoSitter: Ontologia Aplicada à Solução de Problemas em Sistemas Autônomos de Gerenciamento de Banco de Dados"**, orientada pela **Profa. Ana Carolina Brandão Salgado** e aprovada pela Banca Examinadora formada pelos professores:

Prof. Frederico Luiz Gonçalves de Freitas
Centro de Informática / UFPE

Profa. Bernadette Farias Lóscio
Departamento de Computação / UFC

Profa. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 1 de setembro de 2008.

Prof. FRANCISCO DE ASSIS TENÓRIO DE CARVALHO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.