**Pós-Graduação em Ciência da Computação**

# “AutonomousDB: Uma Ferramenta para Propagação Autônoma de Atualizações de Esquemas de Dados”

**Por**

***Arlei José Calazans Moraes***

**Dissertação de Mestrado**

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

Recife, fevereiro de 2009

**UNIVERSIDADE FEDERAL DE PERNAMBUCO**

**CENTRO DE INFORMÁTICA**

**PÓS- GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**ARLEI JOSÉ CALAZANS MORAES**

# "AutonomousDB: Uma Ferramenta para Propagação Autônoma de Atualizações de Esquemas de Dados"

*ESTE TRABALHO FOI APRESENTADO 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: Profª Docteur Ana Carolina Salgado
CO-ORIENTADORA: Profª PhD Patrícia Tedesco

Recife, fevereiro de 2009

Moraes, Arlei José Calazans
AutonomousDB: uma ferramenta para propagação autônoma de atualizações de esquemas de dados / Arlei José Calazans Moraes – Recife : O Autor, 2009.
ix, 104 p.: il., fig., tab., quadros

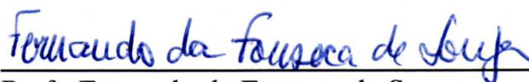
Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da computação, 2009.

Inclui bibliografia.

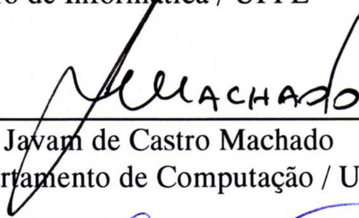
1. Banco de dados. 2. Inteligência artificial. I. Título.

025.4 CDD (22.ed.) MEI2009-018

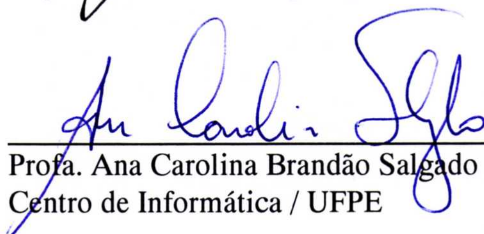
Dissertação de Mestrado apresentada por **Arlei José Calazans Moraes** à Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título "intitulada "**AutonomousDB: Uma Ferramenta para Propagação Autônoma de Atualizações de Esquemas de Dados""**, orientada pela **Profa. Ana Carolina Brandão Salgado** e aprovada pela Banca Examinadora formada pelos professores:



Prof. Fernando da Fonseca de Souza
Centro de Informática / UFPE

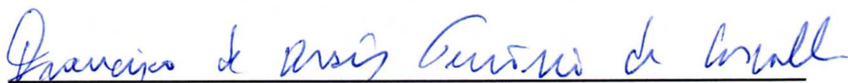


Prof. Javam de Castro Machado
Departamento de Computação / UFC



Profa. Ana Carolina Brandão Salgado
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 17 de fevereiro de 2009.



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

# Agradecimentos


- Em primeiro lugar, agradeço aos meus pais Manoel e Idalina, pelo incessante apoio e pela crença de que "quando temos uma meta, o que era um obstáculo passa a ser uma etapa de um dos planos";
- Às professoras Ana Carolina Salgado e Patrícia Tedesco pela confiança, paciência, suporte e orientação;
- A Chico Bento (Luís Ricardo) pelas brilhantes sugestões para o trabalho e pela constante disponibilidade para tirar dúvidas em muitos momentos;
- À Ana Elizabeth pelo apoio na fase inicial do trabalho;
- A Alexandra Vitório, Amália Chalegre, Rui Teixeira, Adriana Carneiro, Paulo Bastos, Fabiano Rolim, Renata Galante e toda a equipe de DBA do SERPRO Recife que participou do experimento;
- À Talita Rampazzo, pelo apoio e compreensão nas horas difíceis e nas horas fáceis também;
- Ao SERPRO por oferecer apoio ao trabalho e liberação das horas dedicadas ao mestrado;
- A Zé Ramalho, pelas brilhantes letras surrealistas e músicas geniais que acompanharam muitos momentos de produção e reflexão deste trabalho;
- Enfim, agradeço a todos que de alguma forma contribuíram de forma positiva para que a idéia desse trabalho se tornasse realidade. A todos vocês, meu muito obrigado!

# Resumo

Um dos maiores desafios de construir e manter aplicações com longos ciclos de vida é lidar com as inevitáveis mudanças de requisitos que ocorrem com o passar do tempo. Muitas dessas aplicações são dependentes de Sistemas de Gerenciamento de Banco de Dados (SGBD), que geralmente sofrem conseqüências diretas nos esquemas de seus bancos de dados devido a mudanças na realidade que modelam. Essas modificações devem ser muito bem controladas, pois qualquer evolução mal gerenciada pode acarretar sérios problemas de inconsistência de esquemas ou dados. Neste trabalho, propomos uma alternativa para a diminuição dos prejuízos causados pela evolução de esquemas em ambientes distribuídos. Ela utiliza conceitos de Computação Autônoma aplicados a SGBD para propagar atualizações em esquemas replicados pertencentes a um mesmo ambiente, garantindo a consistência entre eles. O protótipo desenvolvido possui uma arquitetura flexível e um Sistema Multi-Agentes (SMA), que executa eventos para realizar a evolução de esquemas nos SGBD alvo. Estes, podem ter vários bancos de dados e serem de diferentes plataformas. Dessa forma, tarefas repetitivas, realizadas pelos Administradores de Banco de Dados (DBA) para garantir a evolução de todas as cópias de um determinado esquema alterado, são eliminadas. Isso não só diminui a ocorrência de erros humanos como possibilita uma maior dedicação dos DBA em atividades de maior importância, tais como: análise, projeto de banco de dados e gerenciamento estratégico dos dados.

**Palavras-chave:** Evolução de Esquemas, Propagação de Atualizações, Sistemas Autônomos

# Abstract

One of the biggest challenge of building and maintaining applications with long life cycles is dealing with the inevitable changes in requirements that occur over time. Many of these applications depend on DBMS that most often suffer direct consequences in their databases schemas due to changes in the reality that they model. This changes must be controlled because any bad managed evolution can cause serious concistency problems for schemas or data. In this work, we propose a new alternative for decreasing schema evolution problems in distributed environments. It uses concepts of Autonomic Computing in DBMS to propagate updates in schemas replicated within the same environment, thus ensuring their consistency. The prototype includes a flexible architecture and a Multi-Agent System, which executes events for updating schemas in the target DBMS. The distributed environment may have several databases in different platforms. Thus, repetitive tasks which are performed by DBA to ensure the evolution in all copies of a specific changed schema are thus eliminated. This will free the DBA to do other tasks that are more important, such as: analysis, database design and strategic management of data, decreasing the probability of human failures.

**Keywords:** Schema Evolution, Update Propagation, Autonomic Systems

# Sumário


**1 Introdução** **1**
- 1.1 Descrição do Problema 1
- 1.2 Motivação 4
- 1.3 Objetivo 4
- 1.4 Metodologia 5
- 1.5 Estrutura do Documento 5

**2 Computação Autônoma em Banco de Dados** **7**
- 2.1 Introdução 7
- 2.2 O que é Computação Autônoma? 8
  - 2.2.1 Histórico 9
  - 2.2.2 Características da Computação Autônoma 10
  - 2.2.3 Considerações arquiteturais 13
  - 2.2.4 Aplicações da Computação Autônoma 17
- 2.3 Computação Autônoma e Gerenciamento de SGBD 18
- 2.4 Aspectos da Computação Autônoma aplicados a SGBD 20
- 2.5 Conclusão 21

**3 Evolução de Esquemas e Propagação de Atualizações** **25**
- 3.1 Introdução 25
- 3.2 Questões sobre Evolução de Esquemas 27
  - 3.2.1 O que é Evolução de Esquemas? 27
  - 3.2.2 Causas da Evolução de Esquemas 28
  - 3.2.3 Modificação, Evolução e Versionamento de Esquemas 31
- 3.3 Projeto de um Sistema de Banco de Dados 32
- 3.4 Requisitos para Evolução de Esquemas 34
- 3.5 Evolução dos Esquemas Conceitual e Lógico 36
  - 3.5.1 Técnicas para tratar a Evolução de Esquemas 37
  - 3.5.2 Estratégias de Propagação nas instâncias 40
  - 3.5.3 Aspectos sobre o uso das Técnicas de Evolução e Estratégias de Propagação 43
- 3.6 Evolução de Esquemas em Sistemas Existentes 43

3.6.1 Iniciativas Acadêmicas 44
3.6.2 Iniciativas Comerciais 48
3.7 Análise Crítica Comparativa 49
3.8 Conclusão 52

**4 AutonomousDB 54**
4.1 Eventos para o AutonomousDB 54
4.2 Especificação da solução 55
4.2.1 Arquitetura geral 55
4.2.2 Modelagem da aplicação 59
4.2.3 Modelo de persistência 70
4.3 Conclusão 75

**5 Uma aplicação real do AutonomousDB: o caso do SERPRO 76**
5.1 O caso do SERPRO 76
5.2 Implementação 81
5.2.1 Camada de Agentes 82
5.2.2 Base de Conhecimento 83
5.3 Experimento Realizado 84
5.4 Resultados Obtidos 85
5.5 Avaliação dos Resultados 87
5.6 Experimento Adicional 90
5.7 Conclusão 91

**6 Conclusões 92**
6.1 Resumo do Trabalho Desenvolvido 92
6.2 Contribuições 93
6.3 Trabalhos Futuros 94

# Lista de Figuras

2.1 Estrutura de um Elemento Autônomo e como ele interage com outros Elementos Autônomos. Fonte: adaptado de [Kephart & Chess, 2003]. 15

3.1 Manutenção de esquemas com base na diminuição do desempenho do banco de dados. Fonte: adaptado de [Sockut & Goldberg, 1979]. 28
3.2 Níveis de abstração do projeto de BD. Fonte: adaptado de [Hick & Hainaut, 2002]. 33
3.3 Estratégia *off-line*, quando os serviços do banco de dados são parados para a realização de atualizações, e a estratégia *on-line*, quando os serviços continuam no ar. Fonte: adaptado de [Camolesi & Traina, 1996]. 37
3.4 Arquitetura para evolução de BD. Fonte: adaptado de [Dominguez et al., 2008]. 46

4.1 Arquitetura do AutonomousDB. 56
4.2 Diagrama de Casos de Uso do AutonomousDB. 60
4.3 Diagrama de Classes de análise para os Casos de uso i) e ii). 63
4.4 Diagrama de seqüência para os Casos de uso i) e ii). 64
4.5 Modelo de Projeto do AutonomousDB. 66
4.6 Fluxo de eventos do AutonomousDB. 71
4.7 Modelo de Dados do AutonomousDB. 74

5.1 Exemplo de como estão estruturados os ambientes de banco de dados do SERPRO. 78

# Lista de Quadros e Tabelas


2.1 Comparativo entre os sistemas atuais e os sistemas autônomos, considerando a presença de aspectos de auto-gerenciamento. Fonte: adaptado de [Kephart & Chess, 2003]. 14

3.1 Comparativo dos sistemas de evolução de esquemas analisados. 52

5.1 Banco D0815 do SERPRO com seus módulos, esquemas, usuários e dependências. 80
5.1 Resultados dos eventos criados e executados pelos DBA no experimento. 87

CAPÍTULO 1

# Introdução

Este capítulo introdutório apresenta as principais motivações para realização deste trabalho, bem como o seu objetivo, a abordagem utilizada para desenvolvê-lo e as suas principais contribuições. Por fim, é apresentada a estrutura como ele está organizado.

## 1.1 Descrição do Problema

O avanço das tecnologias no que se refere ao desenvolvimento de *software*, aliado à diversidade de contextos em que estes podem ser aplicados, torna os sistemas cada vez maiores e mais complexos. Por conseqüência, cresce também a dificuldade para mantê-los. Com sistemas mais complexos e interconectados, garantir segurança, tolerância a falhas e consistência dos esquemas de dados são fatores cada vez mais críticos. Esta situação deve ser tratada com bastante atenção, pois os desenvolvedores não são capazes de antecipar todas as interações entre os componentes de um determinado sistema e, muito menos, identificam a necessidade de alterações nos esquemas de dados. Isso leva à ocorrência de eventos específicos desejados ou indesejados, em tempo de execução. Essa tendência leva os sistemas a um nível de complexidade tão alto que tarefas como instalar, configurar, otimizar, manter (aplicações e esquemas de dados) e integrar ficam cada vez mais difíceis, mesmo para profissionais qualificados. A administração de sistemas computacionais demanda que parte das tarefas, hoje realizadas por administradores humanos, sejam delegadas ou pelo menos auxiliadas por processos automáticos, evitando que o acelerado aumento da complexidade e dinamismo dos sistemas, junto com a diversidade de tecnologias, venha trazer problemas de gerenciamento.

No que concerne ao problema da crescente dificuldade de manutenção, mais especificamente na tarefa de administração de banco de dados, é fato que os sistemas de informação evoluem ao longo do tempo por diferentes razões, seja pelo fato de expansão de

funcionalidades, pela modificação de requisitos técnicos, manutenções corretivas, evolutivas ou adaptativas. Essa evolução afeta os diferentes aspectos das aplicações, que para refletirem a nova realidade do negócio, demandam alterações também nos esquemas do banco de dados utilizado.

Existem várias formas de se organizar um ambiente de banco de dados. Dependendo de cada caso, o SGBD (Sistema de Gerenciamento de Banco de Dados) escolhido pode ser customizado e configurado para tentar atender da melhor forma o que lhe é exigido. O tamanho dos banco de dados, a complexidade do negócio que seus esquemas representam e a organização de seu ambiente (ex: centralizado, distribuído ou *Peer-to-Peer* (P2P)), a evolução de esquemas se torna uma tarefa de administração bem complexa. Essa complexidade agrava-se ainda mais quando o ambiente é composto por vários SGBD de plataformas diferentes e que possuem esquemas replicados em seus bancos de dados. Isso acontece porque se um esquema que está replicado sofrer alguma alteração, ela deve ser refletida em todas as cópias para que a consistência seja mantida. Para que o proceso ocorra de forma satisfatória, também devem ser levadas em conta questões como restrições organizacionais e dependências entre esquemas. Em um contexto como este, se as tarefas de evolução de esquemas e propagação de mudanças forem realizadas por um Administrador de Banco de Dados (DBA) sem nenhum apoio automatizado, a probabilidade de surgirem inconsistências nos esquemas dos bancos de dados e nas instâncias de dados aumenta consideravelmente.

Imagine o seguinte cenário: duas aplicações têm esquemas de dados distintos, cada um devidamente criado em usuários [1] também distintos, pertencentes a um mesmo banco de dados. Devido a uma decisão de projeto, a segunda aplicação utiliza o esquema de dados da primeira para funcionar corretamente. Por este motivo, o esquema de dados da primeira aplicação foi replicado no usuário utilizado pela segunda aplicação. Se ocorrer qualquer modificação no esquema de dados da primeira aplicação, todas as cópias deste esquema no ambiente deverão sofrer a mesma modificação. Isso independente das cópias pertencerem a usuários de um mesmo banco de dados.

Em nosso exemplo, propagar manualmente uma atualização de um esquema de dados replicado em dois usuários pertencentes a um mesmo banco de dados, pode não parecer uma tarefa complicada. Porém, imagine propagar uma atualização em um am-

[1] Entendemos usuário como uma conta de banco de dados e não como um usuário humano do SGBD.

biente onde existe um grande número de usuários, que podem estar criados em diversos bancos de dados com plataformas diferentes. Neste contexto, o trabalho do DBA torna-se bem mais complexo. Ele, sem perceber, pode deixar de atualizar algum esquema ou inserir erros, principalmente se em cada esquema de dados atualizado, for necessário reescrever *procedures*, *packages* e *triggers* de banco. Realizando este trabalho manualmente, o DBA também gera um grande montante de trabalho repetitivo, tendo que efetuar as mesmas atualizações em vários usuários dos bancos de dados.

Para suportar a diversidade de aplicações com as mais diferentes características hoje existentes, os SGBD tornaram-se ferramentas bastante complexas. Eles são dotados de um grande número de parâmetros que precisam ser ajustados e monitorados cuidadosamente, além de terem os esquemas de seus bancos de dados em contínua evolução para corresponder à realidade das aplicações. São ferramentas difíceis e caras de administrar. Devido aos fatores citados, os administradores humanos tornam-se mais susceptíveis a falhas e classificam os SGBD na categoria dos sistemas com forte necessidade para o desenvolvimento de características de autonomia.

As empresas têm custos elevadíssimos com a qualificação de corpo funcional para administrar SGBD, principalmente quando eles atendem a várias aplicações complexas que se integram e possuem bancos de dados gigantescos (na ordem de milhões de registros), dando suporte a serviços essenciais de domínios nada triviais. Evitar trabalho repetitivo e abstrair questões operacionais dos SGBD num contexto como esse, facilitaria bastante o trabalho dos DBA, economizando tempo e conseqüentemente liberando-os para se dedicarem a tarefas mais importantes para o negócio das empresas. Além disso, os custos com a administração de SGBD diminuiriam consideravelmente.

Os objetos de estudo dessa pesquisa são os assuntos e tecnologias destinados a: promoção de autonomia a SGBD, evolução de esquemas de dados, propagação de atualizações, agentes inteligentes e sistemas multi-agentes. Serão estudadas as principais características da Computação Autônoma e como elas podem ser usadas para trazer um certo nível de autonomia na evolução de esquemas e propagação de mudanças. Serão apresentados o processo e os resultados do desenvolvimento de uma nova solução, AutonomousDB. Este é fruto de estudos nas áreas citadas e ameniza um problema pertinente à administração de banco de dados: a evolução de esquemas replicados dentro de ambientes distribuídos e heterogêneos.

## 1.2 Motivação

Existem trabalhos acadêmicos e iniciativas comerciais, cujos objetivos são voltados ao desenvolvimento de aplicações fornecedoras de autonomia a SGBD, principalmente no que diz respeito a auto-ajuste, análise em tempo real de dados em alteração, auto-reparação, tecnologias em *grid* e suporte à integração da informação. No entanto, não encontramos nenhum trabalho que formalize o processo de evolução de esquemas. Dada a larga abrangência e as múltiplas possibilidades de abordagens sobre o tema, verificamos uma grande variedade de investigações desenvolvidas. Dentre todos os trabalhos levantados, nenhum segue o direcionamento de prover autonomia a SGBD através de sistemas multi-agentes no tocante a evolução de esquemas e propagação de mudanças. Todas as propostas exploravam os assuntos independentemente ou com outras perspectivas.

A principal motivação desta pesquisa é minimizar problemas com a administração de banco de dados no tocante à evolução de esquemas, através da construção de uma ferramenta que promove autonomia na propagação de atualizações de esquemas replicados em ambientes distribuídos e heterogêneos. Nesta ferramenta, dado o escopo do problema, será utilizada uma abordagem relativamente recente, a Computação Autônoma, evitando o trabalho repetitivo e erros humanos.

## 1.3 Objetivo

O objetivo desta pesquisa é construir uma solução baseada em sistemas multi-agentes capaz de prover autonomia na tarefa de administração de banco de dados. Essa autonomia diz respeito à tarefa de evolução e propagação de atualizações de esquemas replicados em ambientes distribuídos e heterogêneos. Dessa forma, evitamos que o DBA tenha que atualizar manualmente todas as cópias de um esquema de dados que sofreu alteração. Isso eliminaria o trabalho repetitivo e diminuiria a intervenção humana, reduzindo falhas. Além disso, autonomia na administração de banco de dados, libera os DBA de tarefas operacionais repetitivas para que possam se ater em tarefas mais importantes para o negócio das empresas, como: o planejamento estratégico da gestão dos dados.

## 1.4 Metodologia

Para alcançar os objetivos descritos na Seção 1.3, serão executados os seguintes passos:

1) Aprofundar os conhecimentos acerca de sistemas multi-agentes, técnicas para evolução de esquemas e propagação de atualização em sistemas heterogêneos, identificando trabalhos que utilizam agentes inteligentes para prover autonomia em banco de dados. Feito isso, o próximo passo é fazer um cruzamento dos conhecimentos adquiridos, tentando convergir para a possibilidade de se ter um sistema multi-agentes que ofereça um serviço de evolução e propagação de atualizações de esquemas em um ambiente de banco de dados distribuído e heterogêneo, com um determinado nível de autonomia. O sistema seria capaz de atualizar vários esquemas replicados em diversos bancos de dados, mantendo assim a consistência entre eles, sendo esse processo transparente para o DBA;
2) Especificar uma solução baseada nos estudos feitos no tópico acima, descrevendo todo o processo de desenvolvimento;
3) Implementar um protótipo da solução especificada e aplicá-lo no Serviço Federal de Processamento de Dados (SERPRO), coletando informações do experimento através de um questionário respondido pelos DBA participantes; e
4) Avaliar o desempenho do protótipo, analisar as respostas do questionário e descrever os principais resultados obtidos.

## 1.5 Estrutura do Documento

Além deste capítulo, a dissertação está estruturada como segue:

- **Capítulo 2: Computação Autônoma em Banco de Dados**
  Este capítulo aborda os principais conceitos da Computação Autônoma e Inteligência Artificial aplicados no contexto de banco de dados, com a explicitação das contribuições da Computação Autônoma e aspectos dela aplicados a SGBD;

- **Capítulo 3: Evolução de Esquemas e Propagação de Atualizações**
  Este capítulo descreve os principais conceitos sobre Evolução de Esquemas, apre-

sentando uma contextualização e necessidade do seu uso, bem como informações sobre como é desenvolvido um projeto de Sistema de Banco de Dados. Também são apresentados os requisitos básicos a serem atendidos para uma evolução de esquemas satisfatória, uma descrição das principais técnicas e estratégias utilizadas e uma análise de alguns trabalhos cujos objetivos estão relacionados ao problema principal da realização da evolução de esquemas com algum suporte à propagação de mudanças/atualizações;

- **Capítulo 4: AutonomousDB**
  Este capítulo relata a etapa da especificação da solução proposta provedora de autonomia para SGBD, seus aspectos, arquitetura, modelos de análise e projeto, requisitos, diagramas e ambiente em que atua;

- **Capítulo 5: Uma aplicação real do AutonomousDB: o caso do SERPRO**
  Este capítulo descreve uma implementação do AutonomousDB, demonstrando tecnologias utilizadas e uma aplicação real da solução especificada no SERPRO (estudo de caso), descrevendo os detalhes do experimento e a análise dos principais resultados obtidos; e

- **Capítulo 6: Conclusões**
  Este último capítulo apresenta uma síntese do trabalho desenvolvido e aponta suas principais contribuições. Também são relatadas as possibilidades de pesquisas futuras que tenham como referência o AutonomousDB.

CAPÍTULO 2

# Computação Autônoma em Banco de Dados

Este capítulo descreve os principais conceitos da Computação Autônoma, suas características e os motivos principais de sua utilização na construção de soluções de apoio a Sistemas de Gerenciamento de Banco de Dados (SGBD). O entendimento dos conceitos apresentados a seguir são de importância fundamental para a compreensão do restante do trabalho. A Seção 2.1 apresenta uma breve introdução sobre Computação Autônoma. A Seção 2.2 aborda os principais conceitos, um breve histórico, suas características, algumas considerações arquiteturais sobre sistemas autônomos e suas principais aplicações na indústria de Tecnologia da Informação (TI). As Seções 2.3 e 2.4 descrevem a relação do uso da Computação Autônoma no gerenciamento de SGBD e seus aspectos específicos. Por fim, a Seção 2.5 traz algumas considerações importantes para finalizar o capítulo.

## 2.1 Introdução

Nos últimos anos, diversas transformações vêm acontecendo na indústria da TI em um processo de globalização jamais visto, que deu origem a uma crescente demanda por sistemas computacionais cada vez mais poderosos e interligados. Essa necessidade surgiu do interesse de tornar os indivíduos e as empresas mais produtivas, através da automatização de tarefas e processos-chave. Porém, um fato de suma importância que deve ser levado em conta é que a evolução através da automatização traz como conseqüência uma complexidade inevitável aos sistemas computacionais.

Diversos fatores introduziram novos níveis de complexidade aos sistemas, tais como: o avanço das tecnologias de banco de dados e Internet têm aumentado significativamente a capacidade de armazenamento de dados estruturados e não estruturados. Os sistemas deixaram de ser essencialmente centralizados para serem distribuídos e heterogêneos.

As redes de computadores passaram a interligar cada vez mais esses novos tipos de sistemas. As inovações nas linguagens de programação têm aumentado o tamanho e a complexidade deles [Ganek & Corbi, 2003].

Com o surgimento de sistemas computacionais mais complexos e interligados, a atividade de administrá-los torna-se cada vez mais difícil, mesmo para profissionais altamente qualificados, além de que o custo com esse tipo de profissional é alto e pode tornar-se um empecilho no desenvolvimento da indústria da TI. Outro problema encontrado são os erros humanos. Em sistemas complexos, pessoas são mais suceptíveis a falhas, aumentando a probabilidade de inclusão de erros nos sistemas. Essas dificuldades aliadas à velocidade da evolução tecnológica,"podem levar a indústria de TI a uma crise de gerenciamento, esse é o mote decisivo que dá à Computação Autônoma a importância e atenção das comunidades de pesquisa e desenvolvimento comerciais e acadêmicas atuais" [Maciel, 2007].

## 2.2 O que é Computação Autônoma?

Na literatura existem três definições que expressam bem o propósito da autonomia em sistemas computacionais modernos. A primeira delas diz que Computação Autônoma é uma abordagem de construção de *software* que propõe a utilização de tecnologia para gerenciar tecnologia [IBM Autonomic Blueprint, 2006]. A segunda define Computação Autônoma como a evolução lógica de tendências passadas para resolver o crescente problema da complexidade dos sistemas e ambientes de computação distribuída [Ganek & Corbi, 2003]. E por fim, Computação Autônoma é o conjunto de características capaz de permitir que sistemas computacionais possam gerenciar a si mesmos, resultando em objetivos de alto nível definidos por seus administradores [Horn, 2001]. Por essas definições, vemos que o foco da Computação Autônoma está em resolver problemas causados pela crescente complexidade dos sistemas. Através dela, é possível delegar atividades realizadas por administradores humanos a processos automáticos, diminuindo dessa forma a intervenção humana, e conseqüentemente, eliminando possíveis erros. O ganho direto com isso é a economia de tempo dos administradores, liberando-os para atividades mais importantes; diminuição de custos e aumento da agilidade do processo.

A construção de *software* vista pela ótica da Computação Autônoma prega que os componentes de um sistema sejam desenvolvidos incorporando mecanismos de auto-gerenciamento, obtendo dessa forma um nível relativamente alto de independência em relação aos administradores humanos na realização de tarefas como: configuração, proteção, otimização e recuperação. Dessa forma, evitamos que o aumento da complexidade de gerenciamento e diversidade de tecnologias venha a complicar o desenvolvimento do ambiente de TI atual e futuro.

### 2.2.1 Histórico

Paul Horn, diretor de pesquisa da IBM, apresentou em seu trabalho "*Autonomic Computing: IBM's Perspective on The Sate of Information Technology*" [Horn, 2001] um manifesto, onde alertou à comunidade de TI sobre o crescente problema da complexidade das tecnologias e a real dificuldade em gerenciá-las. Ele relatou que não há como criar recursos de monitoramento, acompanhamento e intervenção para os novos sistemas na mesma velocidade, diversidade e quantidade em que surgem. Todos os fatores que contribuem para o aumento da complexidade dos sistemas, descritos na Seção 2.1, não são devidamente acompanhados. Não há a preocupação de projetar, monitorar e gerenciar individualmente componentes que vão inteirar sistemas computacionais. Diante dessa situação, a probabilidade de uma crise de gerenciamento devido à complexidade é deveras alta, tornando clara a necessidade de mudar a forma como o *software* e sistemas são desenvolvidos, sob pena de se tornar impraticável gerar recursos para administrá-los.

Segundo Horn (2001), é necessário projetarmos e construirmos sistemas auto-gerenciáveis capazes de se ajustar a diversas circunstâncias que sejam escaláveis e lidem de forma eficiente com situações críticas. Esses sistemas, chamados autônomos, devem ser pró-ativos às suas necessidades, permitindo os DBA focarem no objetivo que querem atingir, evitando-os de despender tempo configurando os sistemas, dizendo como eles devem agir para que os objetivos sejam atingidos.

As idéias apresentadas no manifesto de Paul Horn tiveram um impacto significativo na comunidade de TI. Um fato que demonstra essa afirmativa é que cada vez mais encontramos características de autonomia tanto em sistemas novos em desenvolvimento,

como em sistemas já lançados no mercado e em produção. Absorvida a necessidade da autonomia, os grandes produtores de *software* mundiais começaram a investir nessa idéia. Iniciativas como a criação de laboratórios específicos, novas divisões e apoio a projetos acadêmicos já são realidade e, por este último, podemos perceber o número cada vez maior de trabalhos publicados no campo da Computação Autônoma. A IBM, por exemplo, reorganizou toda a sua divisão de pesquisa, com 3.200 profissionais e orçamento anual de cerca de 5 bilhões de dólares, em torno da meta de dar autonomia aos sistemas de informação e apoiar projetos acadêmicos que possam contribuir para a sua consecução. Já conseguiu apoio de empresas como a Microsoft e a Sun [Marques, 2002].

### 2.2.2 Características da Computação Autônoma

O desafio de automatizar o gerenciamento de recursos computacionais não é um problema recente. No passado, a maioria dos sistemas era em sua essência não distribuída, trabalhava com um volume menor de dados para armazenamento, tinha um nível baixo ou nenhum de heterogeneidade e interconexão. Construir recursos para gerenciar esse tipo de ambiente era relativamente mais simples que nos dias de hoje. Com o avanço das tecnologias durante décadas, os sistemas de controle, compartilhamento de recursos, tempo real, dentre muitos outros, foram evoluindo e se tornando mais complexos e interconectados. Os sistemas e componentes de *software* responsáveis pelo gerenciamento desse ambiente também foram evoluindo constantemente para acompanhar e lidar de maneira eficiente com o aumento da complexidade dos sistemas gerenciados. A Computação Autônoma pode ser vista como uma nova tendência para resolver, ou pelo menos amenizar o problema da complexidade e da computação distribuída nos ambientes atuais.

Horn (2001) descreve em seu manifesto esse grande desafio de contornar a complexidade de gerenciamento nos sistemas e propõe oito elementos-chave que permitem automatizar atividades de instalação, configuração, otimização, manutenção, integração e gerência de conflitos, traduzindo objetivos de alto nível dos administradores. Esses elementos-chave são críticos e cada vez mais desejados nos sistemas atuais, uma vez que a complexidade e o volume de tarefas a serem realizadas para administrar os sistemas computacionais só tendem a aumentar.

Segundo Horn (2001), os oito elementos-chave que caracterizam os Sistemas de Computação Autônoma são:

1. Conhecer a si próprio, ou seja, ter uma identidade de sistema, conhecer seus componentes, estados, funções e interações com o meio externo;
2. Possuir capacidade de se configurar e reconfigurar;
3. Realizar constante auto-otimização;
4. Ser capaz de se reparar;
5. Ser capaz de se proteger;
6. Usar auto-adaptação para melhor interagir com o meio-ambiente e sistemas vizinhos;
7. Ser uma solução não-proprietária, aberta e baseada em padrões; e
8. Prever e antecipar recursos otimizados necessários, mantendo a complexidade oculta.

Horn (2001) diz que todo sistema para se caracterizar como um Sistema Autônomo **deve incorporar** todos os oito elementos-chave descritos acima. Ao definir Computação Autônoma (ver Seção 2.2) e os elementos-chave, ele faz uma analogia ao corpo humano e seu sistema nervoso autônomo, onde os órgãos de maneira individual "têm conhecimento" de suas próprias funções e de como executá-las. Cada órgão gerencia a si mesmo em perfeita harmonia com os demais, sem a necessidade de conscientemente emitirmos estímulos para controlar suas funções de baixo nível ou vitais. Por exemplo: o sistema nervoso autônomo controla a freqüência de nossos batimentos cardíacos e tenta manter a temperatura do nosso corpo constante, sem que precisemos explicitamente e conscientemente fazê-los.

Continuando com a analogia, podemos notar que em todos os níveis de granularidade da cadeia biológica de formação do corpo humano, desde as células de forma individual até sua agregação para formar os tecidos, órgãos e organismos, existem unidades que se combinam formando subsistemas especializados que também possuem regras e mecanismos próprios de auto-regulação. De forma similar, a essência da característica de auto-gerenciamento dos sistemas no ambiente de TI são pequenas unidades autônomas, chamadas de elementos autônomos. Elas também se combinam formando subsistemas, que são responsáveis por cuidar da grande massa de gerenciamento operacional existente no ambiente de TI, automatizando uma tarefa árdua, dinamizando e

permitindo as organizações focarem em cenários mais estratégicos [Maciel, 2007].

A essência da Computação Autônoma é a característica de auto-gerenciamento. Sua intenção é de livrar os administradores de detalhes operacionais e de manutenção dos sistemas, maximizando dessa forma o desempenho de uma forma geral. Sistemas computacionais podem se auto-gerenciar através de objetivos de alto nível, regras e políticas definidas pelos administradores. Para que seja possível a incorporação dos oito elementos-chave nos sistemas de computação autônoma para caracterizar o auto-gerenciamento, quatro aspectos são de fundamental importância: auto-configu-ração, auto-otimização, auto-reparação e auto-proteção [Kephart & Chess, 2003]. O contexto e sentido desses quatro aspectos não devem ficar restritos a sua definição, mas sim amadurecer e evoluir constantemente de acordo com novas experiências e necessidades no ambiente da indústria de desenvolvimento de *software*, como segue:

- **Auto-configuração** - os sistemas se adaptam automaticamente a mudanças em ambientes dinâmicos, sem a necessidade de adaptações forçadas. Esse aspecto permite que novas características sejam adicionadas à infra-estrutura de TI (ex. adição de um novo *software*) com o mínimo impacto nos serviços fornecidos e com a mínima intervenção humana possível. Algumas capacidades como: *plug-and-play*, *wizards* de configuração e tecnologias *wireless* ajudam a diminuir a intervenção humana. O objetivo mais importante da auto-configuração ocorre no nível estratégico das empresas, no qual qualquer recurso computacional adicionado à infra-estrutura de TI deve ser ajustado de maneira a maximizar a qualidade dos serviços. Para esse ajuste, devem ser levadas em conta as políticas organizacionais vigentes e o alinhamento dos serviços de TI com os objetivos de negócio;

- **Auto-otimização** - os sistemas buscam de forma permanente, dentro do ambiente em que se encontram, maneiras de melhorar suas operações, identificar e aprender com novas oportunidades para, dessa forma, aumentar sua própria eficiência, seja relativo ao desempenho ou ao custo. Eles buscam maximizar a utilização de recursos, adaptando-se às necessidades dos usuários sem precisar de intervenção humana. Esse aspecto é muito importante no contexto de sistemas heterogêneos, pois através dele é possível alcançar uma melhor distribuição e balanceamento de cargas de trabalho. Além disso, as empresas podem otimizar recursos em sua

infra-estrutura de TI, enquanto mantêm a flexibilidade para fazer as melhorias que sejam necessárias;

- **Auto-proteção** - os sistemas detectam, identificam e se protegem de problemas em larga escala, originados de ataques maliciosos ou falhas em cascata. Devem antecipar problemas baseados em *reports* de sensores e tomar as devidas providências para evitá-los ou atenuá-los. Além disso, possuem a habilidade de definir e gerenciar o acesso de usuários a todos os recursos computacionais da empresa para evitar acessos não autorizados, detectar intrusões, comportamentos maliciosos ou qualquer falha que comprometa a segurança do sistema como um todo. Eles ainda devem emitir alertas e prever identificação e autorização de usuários em contextos de acessos diferentes (ex. métodos de acesso *single sign-on*), reduzindo a sobrecarga de administradores humanos; e

- **Auto-reparação** - os sistemas detectam, fazem diagnósticos e reparos de problemas localizados resultantes de falhas no *software* ou *hardware*. Eles devem se recuperar e isolar falhas, utilizando conhecimento sobre a configuração do sistema, prevenindo impactos que possam causar descontinuidade de serviços, buscando assim maximizar a disponibilidade dos serviços oferecidos pela empresa.

Diversos problemas são enfrentados devido à falta de características autônomas nos sistemas atuais no que diz respeito à administração. O Quadro 2.1 faz um comparativo entre as características principais nos sistemas comuns e autônomos. Todos os quatro aspectos explanados anteriormente são propriedades emergentes de uma arquitetura geral para componentes autônomos; e conseqüentemente, sistemas autônomos, como veremos na Seção 2.2.3.

### 2.2.3 Considerações arquiteturais

Antes de fazer considerações sobre como está estruturado um sistema autônomo, vamos primeiro definir o que é um elemento autônomo. Segundo Kephart e Chess (2003), elementos autônomos são unidades básicas dos sistemas autônomos capazes de gerenciar seu comportamento e relações com outros elementos de acordo com políticas de alto nível estabelecidas por humanos ou outros elementos. Possuem recursos próprios e

**Quadro 2.1** Comparativo entre os sistemas atuais e os sistemas autônomos, considerando a presença de aspectos de auto-gerenciamento. Fonte: adaptado de [Kephart & Chess, 2003].

| Aspecto | Computação Atual | Computação Autônoma |
|---|---|---|
| Auto-configuração | Grandes centrais de dados das corporações vêm de inúmeros fornecedores e plataformas diferentes. Instalar, configurar e integrar tudo isso consome muito tempo e aumenta a margem de erro. | Configurações automáticas de componentes e sistemas seguem políticas de alto-nível. Sistemas se ajustam automática e continuamente a novas mudanças. |
| Auto-otimização | SGBD têm centenas de parâmetros de sintonia não lineares e modificados manualmente com esse número crescendo a cada nova versão. | Componentes e sistemas procuram continuamente oportunidades de aumentar seu desempenho e eficácia. |
| Auto-reparação | Determinação de problemas em sistemas grandes e complexos pode ocupar uma equipe grande de programadores por várias semanas. | Sistemas detectam, fazem diagnósticos e reparam automaticamente problemas localizados de *software* e *hardware*. |
| Auto-proteção | Detecção e recuperação de ataques de segurança e falhas em cascata são feitas manualmente. | Sistemas se defendem automaticamente contra ataques maliciosos ou falhas em cascata. Para isso, usam avisos prévios para antecipar e prevenir grandes falhas no sistema. |

fornecem serviços necessários para o funcionamento de outros elementos autônomos.

Sistemas autônomos podem ser vistos como coleções interativas de elementos autônomos, os quais consistem em um ou mais elementos gerenciados acoplados a um gerente autônomo que os controla e representa. Elementos gerenciados podem ser recursos de *hardware* como impressoras, discos rígidos, *drives* ou podem ser recursos de *software* como banco de dados, sistemas legados e ferramentas de desenvolvimento. Eles podem ser adaptados por meio de mecanismos físicos ou *middleware*, de forma a permitir que os gerentes autônomos possam monitorá-los e controlá-los. “Devido a suas características, os sistemas autônomos têm sido considerados nas recentes iniciativas de

prover autonomia para atividades de gerenciamento de SGBD" [de Albuquerque, 2007].

Elementos autônomos gerenciam seu estado interno e comportamento através de objetivos embutidos, ou seja, cada elemento tem seus objetivos específicos e fornecem determinados serviços de forma individual. Como eles trabalham em coleções interativas, um elemento precisa do serviço de outro ou outros, para em conjunto atingir objetivos gerais de alto nível. Dessa forma cada elemento, quando solicitado, tenta atingir seus objetivos específicos contribuindo para objetivos mais gerais, dividindo também falhas quando ocorrem.

A arquitetura básica para um elemento autônomo é composta por um gerente de autonomia acoplado a nenhum ou vários elementos gerenciados. O gerente de autonomia recolhe requisições, através de sensores, e efetua ações específicas para prover seus serviços através de atuadores, que servem de interfaces de comunicação entre o gerente de autonomia e o elemento gerenciado. A Figura 2.1 ilustra a arquitetura básica descrita.

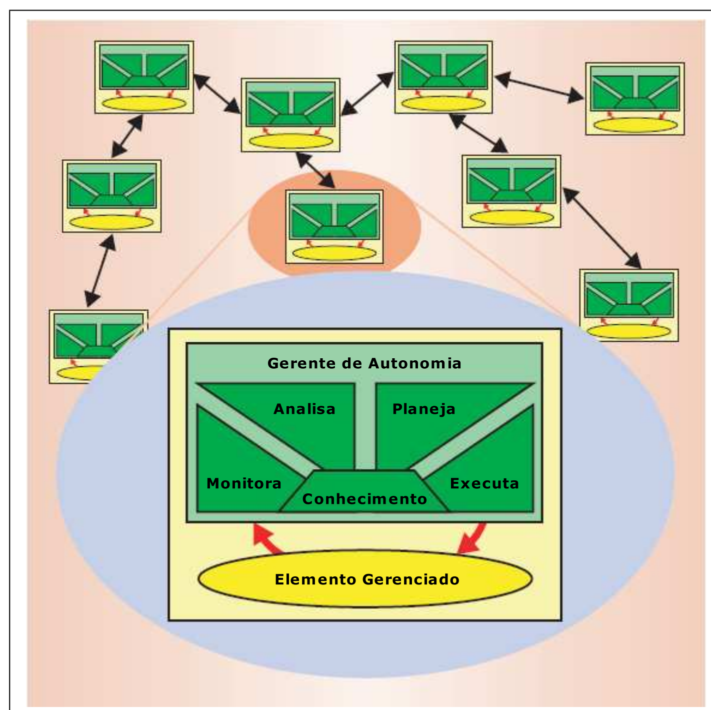



**Figura 2.1** Estrutura de um Elemento Autônomo e como ele interage com outros Elementos Autônomos. Fonte: adaptado de [Kephart & Chess, 2003].

Quando o gerente de autonomia recebe uma requisição de serviço, esta passa por um laço de controle que é composto pelas atividades de monitoração, análise, planeja-

mento e execução. O fluxo de informações se dá como segue: o elemento gerenciado é monitorado através de sensores. No momento em que a requisição é feita ou um evento é identificado, os sensores repassam a informação para análise, que vai verificar a natureza da requisição ou evento, dependências, impactos e que ações serão aplicadas para atender a demanda. Logo após a análise, é planejado como as ações serão aplicadas da melhor forma, minimizando impactos e indisponibilidades de serviços, para só assim serem executadas através dos atuadores. Todo esse laço de controle é intermediado pelo conhecimento interno do gerente de autonomia, que dita as políticas de alto nível e tem informações históricas sobre os resultados de requisições passadas. Isso ajuda nas tomadas de decisões e na maximização do desempenho do sistema como um todo.

A combinação de elementos autônomos para formar sistemas autônomos deve fornecer uma estrutura capaz de operar em ambientes distribuídos e orientados a serviço, pois os elementos autônomos interagem com outros para prover ou consumir serviços, seguindo especificações de alto nível definidas por administradores. As interfaces de comunicação dos elementos autônomos devem ser bem definidas. Elas são importantes para facilitar a comunicação entre os componentes dos sistemas autônomos, seja comunicação intra-sistema ou inter-sistemas. É aconselhável também o uso de padrões abertos e bem conhecidos.

Muito se tem avançado na busca de definição de padrões arquiteturais para computação autônoma. Um fato disso é que existem diversas outras arquiteturas baseadas e também independentes da arquitetura básica descrita, bem como outros padrões. Alguns exemplos são: a Arquitetura de Referência da IBM [IBM Autonomic Blueprint, 2006], a Arquitetura Baseada em Redes de Autômatos [Koehler et al., 2003] também da IBM, a QADPZ (*Quite Advanced Distributed Parallel Zystem*) [Constantinescu, 2003], arquitetura que busca a utilização de computadores em ambientes heterogêneos para execução distribuída de aplicações, a *Legacy Systems* [Fuad & Oudshoorn, 2006], arquitetura para fornecer características autônomas a sistemas legados, o padrão IETF [1](*Internet Engineering Task Force*), que busca desenvolver uma linguagem de definição de políticas e protocolos de gerenciamento de rede, dentre outros.

Elementos autônomos têm uma estreita relação com agentes inteligentes. Em seu livro *"Artificial Intelligence: A Modern Approach"*, Russell e Norvig (2003) dizem

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

que as bases teóricas da Computação Autônoma estão em sua maioria ligadas a técnicas da Inteligência Artificial. Realmente, analisando elementos autônomos, vemos que eles têm ciclos de vida complexos, lidam com atividades em múltiplos *threads*, estão em constante interação com o ambiente em que estão inseridos, além de terem como características marcantes autonomia, pró-atividade e interação baseada em objetivos. Fazendo um paralelo, vemos que todas as características citadas anteriormente, são também características básicas de Agentes Inteligentes de *Software*. "Enxergando elementos autônomos como agentes inteligentes e sistemas autônomos como sistemas multi-agentes, torna-se claro que os conceitos de arquitetura multi-agente são importantes para a Computação Autônoma." [Kephart & Chess, 2003].

### 2.2.4 Aplicações da Computação Autônoma

As aplicações da Computação Autônoma são variadas. Elas ocorrem principalmente onde são necessários recursos computacionais de auto-gerenciamento. E também em sistemas que demandam auto-adaptação a condições diversas de contexto operacional ou recursos de tempo real. Como exemplos temos: *software* crítico e de alta disponibilidade de recursos, *software* de resposta dinâmica às prioridades do negócio, ferramentas de gerenciamento de banco de dados que fazem análise de estatísticas e aprendem pelo histórico de desempenho dos sistemas, *software* de auditoria e segurança, gerenciamento de recursos computacionais distribuídos, entre vários outros.

Como a Computação Autônoma é uma abordagem relativamente recente de construção de *software*, sua aplicação na indústria de TI ainda requer um certo amadurecimento. Na indústria, a maior parte da infra-estrutura de TI é composta por plataformas e componentes heterogêneos. Na maioria das vezes, esse ambiente integra produtos de muitos e diversos fornecedores, o que dificulta a integração e o auto-gerenciamento de forma global. A quantidade e diversidade de sistemas existentes no ambiente da indústria de TI precisam de mais padronização. Faz-se necessária a introdução de uma abordagem uniforme para configuração dinâmica, operação, manipulação, armazenamento e recuperação de dados. A uniformidade permite a troca cada vez mais fácil de informações entre os sistemas que se integram, além de facilitar o controle de informações, tornando possível a criação de bases contendo dados de colaboração e de comportamentos autônomos entre os sistemas heterogêneos [Ganek & Corbi, 2003].

## 2.3 Computação Autônoma e Gerenciamento de SGBD

Sistemas de Gerenciamendo de Banco de Dados (SGBD) são componentes de importância fundamental para os sistemas de informação da atualidade, pois são os maiores responsáveis pelo alto desempenho, disponibilidade, confiabilidade e forte segurança dos sistemas que fazem uso dos seus recursos. São utilizados por uma grande diversidade de aplicações, cada qual com suas características próprias, nos mais variados contextos, como exemplo: comércio eletrônico, governamental, *Data Warehousing*, banco de dados P2P, distribuídos e multimídia. Essa diversidade faz com que os SGBD se tornem ferramentas complexas, acumulando um grande número de características que precisam ser configuradas, ajustadas e monitoradas para maximizarem seu potencial de acordo com o ambiente em que se encontram. SGBD comerciais bem populares, tais como: Oracle [2], DB2 [3] e SQL Server [4] possuem vários parâmetros de configuração que controlam seus funcionamentos de maneira adequada ou aperfeiçoada em função de determinadas condições. A respeito desse fato, poucos profissionais especializados detêm o conhecimento específico de como realizar sintonia para desempenho ou ajustes finos [Maciel, 2007].

Administrar SGBD não é uma tarefa fácil, principalmente quando são aplicados a ambientes heterogêneos de domínios não triviais. Nestas condições, SGBD demandam profissionais altamente qualificados e experientes na tarefa de gerenciá-los. Profissionais neste nível são escassos no mercado e pela lei da oferta e procura, o custo em contratá-los é altíssimo. Muitas empresas optam por qualificar administradores através de treinamentos e cursos, o que também é custoso. Nesta opção, ainda existe o risco da empresa depois de todo o investimento na formação dos administradores, perder estes profissionais para outras empresas que oferecerem condições melhores.

Tarefas realizadas no dia-a-dia de trabalho dos administradores de SGBD, como a realização de cargas de dados, planejamento de recuperação e alta disponibilidade, manutenções cotidianas, provisão de recursos, definição de políticas de segurança, otimização de consultas SQL; e principalmente, evolução contínua de esquemas de dados e projeto, só evidenciam ainda mais a referida complexidade de gerenciamento. Devido

[2] http://www.oracle.com/technology/products/database/oracle10g

[3] http://www-306.ibm.com/software/data/db2

[4] http://www.microsoft.com/brasil/servidores/sql/default.mspx

ao contexto descrito, podemos classificar SGBD como ferramentas com forte necessidade de desenvolvimento de características de autonomia.

Desde a identificação da necessidade de automatizar tarefas e gerar algum grau de independência em relação à intervenção humana, os principais e mais populares SGBD de mercado (Oracle, DB2, SQL Server), já proporcionaram avanços significativos no campo da automatização de tarefas. Os avanços aumentam a cada nova versão lançada. Uma prova disso são as tendências em relação aos esforços de desenvolvimento de características de autonomia [Lightstone et al., 2003]:

- Mudança para sistemas que realizem auto-ajuste fino contínuo e adaptação a cargas de trabalho em detrimento de sistema ajustados estaticamente ou por intervenção humana;
- Desenvolvimento de sistemas que ofereçam suporte a análises em tempo real de dados em constante alteração;
- Criação de novas classes de funções que ofereçam suporte à integração de informação de sistemas distribuídos e heterogêneos;
- Criação de tecnologias em *grid* para bancos de dados;
- Substituição gradual de armazenamento local de dados por armazenamento atrelado à rede e disponibilizado para múltiplos ambientes;
- Reconhecimento que paralisações não são aceitáveis e disponibilidade e continuidade de serviço são funções básicas de qualquer SGBD;
- Desenvolvimento de sistemas auto-reparáveis; e
- Reconhecimento renovado da importância da segurança de dados em SGBD.

Esses esforços vêm incorporando nos sistemas de gerenciamento características de autonomia, fornecendo-lhes a capacidade de operarem com um certo nível de independência dos administradores humanos no ambiente em que se encontram. Isso ocorre principalmente no tocante a eventos que acontecem de forma repetitiva ou semelhante no ambiente gerenciado, pois muitos desses sistemas utilizam bases de dados (repositórios) e motores de inferência para guardar informações de eventos ocorridos e realizar

análises estatísticas para aprendizado. A utilização de análises históricas e tempo de resposta de transações ajuda de forma eficiente na identificação de pontos críticos de falhas e na melhor elaboração de planos para execução de consultas SQL.

Maciel (2007) traz em seu trabalho vários exemplos (descritivos) de avanços proporcionados pelos três maiores fabricantes de SGBD do mercado: IBM com o DB2, Microsoft com o SQL Server, Oracle com o Oracle 10g e os projetos de pesquisa para seus produtos. Traz também iniciativas de código-fonte aberto ou acadêmicas como: Projeto SMART da IBM [Lohman & Lightstone, 2002], Projeto Microsoft AutoAdmin [Autoadmin, 2007], inovações de auto-gerenciamento do Oracle 10g, Sistema DBSitter (UFPE) [Carneiro et al., 2004], Sistema ADAM (Western Ontario University) [Ramanujam & Capretz, 2005], entre outros.

Na Seção 2.4, falaremos de forma complementar sobre os aspectos da Computação Autônoma focados em SGBD, bem como das estratégias para atingir maiores níveis de autonomia nesse tipo de sistema.

## 2.4 Aspectos da Computação Autônoma aplicados a SGBD

Na Seção 2.2.2, explanamos os quatro aspectos que caracterizam o auto-gerenciamento em sistemas autônomos (auto-configuração, auto-otimização, auto-reparação e auto-proteção). Quando tratamos mais especificamente de SGBD autônomos, temos mais dois aspectos além dos quatro citados que são de importância fundamental, são eles [Elnaffar et al., 2003]:

- **Auto-organização** - é a capacidade de organizar e re-estruturar os dados gravados (tabelas) associados aos dados auxiliares (índices) para otimizar o desempenho; e
- **Auto-inspeção** - é a capacidade de tomar decisões inteligentes pertinentes a todas as características anteriormente descritas, tais como: coletar, armazenar e analisar informações sobre desempenho e carga de trabalho.

Construir um SGBD autônomo do zero não é uma tarefa fácil, pois ele deve levar em conta todos os aspectos de auto-gerenciamento desde sua fase de projeto. Como os mais populares SGBD existentes não foram construídos originalmente para serem

autônomos, é menos complexo fazê-los incorporar características autônomas evoluindo suas versões para se tornarem autônomos que reconstruí-los do zero como autônomos. Baseados nisso, encontramos duas estratégias que vêm sendo empregadas para atingir maiores níveis de autonomia nos SGDB: a reescrita de todo o código fonte dos sistemas, utilizando técnicas que os torne autônomos e a incorporação controlada e gradual de mecanismos de autonomia nos sistemas existentes. A tendência observada pela indústria de TI é que se opte pela segunda estratégia, pois reescrever o código legado já existente dos sistemas, contemplando aspectos autônomos, é de uma complexidade comparável a reconstruí-los do zero.

As evoluções dos SGBD mais importantes existentes no mercado tentam incorporar características de autonomia ao código fonte de seus produtos de forma gradual e incremental. Todas essas modificações e melhoramentos são lançados em conjunto com novas versões. Esta é a forma mais conhecida de evolução dos SGBD advindos do mercado para incorporarem autonomia, pois é a menos custosa, mais prática e de melhor aceitação pelos clientes. SGBD como o DB2 da IBM e o Oracle 10g da Oracle, já incorporam características autônomas como, por exemplo, auto-otimização através de otimizadores de consultas SQL, auto-configuração através de assessores de configuração e auto-reparação através de ferramentas de recuperação. Apesar de possuírem algumas características autônomas, esses SGBD ainda não são considerados totalmente como Sistemas Autônomos de Gerenciamento de Banco de Dados (SAGBD), pois recursos que fornecem características de autonomia, descritas anteriormente, ainda estão sendo criados, estudados e outros melhorados. Até atingirem um patamar confiável, estável e um escopo maior para prover autonomia com qualidade, muitos estudos, pesquisas e esforços ainda precisam ser empregados.

## 2.5 Conclusão

Devido ao rápido crescimento da complexidade dos sistemas computacionais atuais e a conseqüente dificuldade em gerenciá-los e mantê-los, a Computação Autônoma surgiu como uma solução atrativa e de grande aceitação pela indústria da Tecnologia da Informação (TI). Até que todo o ambiente dessa indústria atinja níveis consideráveis de autonomia, capazes de suprir a necessidade existente nos sistemas atuais em produ-

ção e nos que ainda estão por vir, grandes desafios ainda precisam ser superados. A Computação Autônoma ainda tem muito o que amadurecer e evoluir, mas já mostra resultados concretos com o desenvolvimento de características de autonomia como solução para administrar sistemas complexos, interconectados e de alto custo de aquisição e manutenção. SGBD se encaixam perfeitamente nesse grupo, pois possuem todas essas características e conseqüentemente necessitam fortemente de recursos autônomos.

No que diz respeito à Computação Autônoma aplicada a banco de dados, o desenvolvimento de características de autonomia em SGBD significa a diminuição de tarefas repetitivas realizadas pelos DBA, liberando-os para se dedicarem ao desenvolvimento de novas aplicações ou tarefas mais importantes para o negócio da empresa.

Os avanços no desenvolvimento de características autônomas nos SGBD atuais são bastantes significativos e trazem boas perspectivas para o futuro, porém ainda existe muita intervenção humana requerida por esses sistemas para ser reduzida. Podemos elencar os principais aspectos que são responsáveis por consumir muita força de trabalho dos DBA e que ainda não estão presentes da forma como deveriam nos SGBD, são eles [Elnaffar et al., 2003]:

1. Grande dependência de decisões humanas - Muitas tarefas ainda continuam exigindo uma quantidade significativa de entradas, inteligência e tomada de decisão por parte do DBA;
2. Necessidade de Adaptação Dinâmica - O recurso de autonomia *tuning advisors* tem sido muito útil na configuração inicial dos SGBD, porém ele ainda não é eficiente para se adaptar automaticamente às mudanças no ambiente ou às variações da carga de trabalho;
3. Falta de capacidade de redefinir parâmetros on-line - *Tuning* dinâmico requer que todos os recursos e parâmetros dos SGBD sejam reguláveis sem causarem interrupções nos serviços do sistema. Essa capacidade ainda não está presente nos SGBD atuais;
4. Falta de capacidades analíticas - Muitas ações corretivas nos SGBD normalmente ainda dependem da experiência dos DBA. Os SGBD devem ser capazes de aprender com a experiência dos DBA; e dessa forma, construir bases de conhecimentos que podem ser utilizadas para a determinação e resolução de problemas de forma automática;

5. Falta de estratégias inteligentes de manutenção - Os SGBD atuais ainda não têm capacidade de prever qual o melhor momento de realizar, nem quanto tempo irá durar, tarefas como: *rebinding*, gravação de estatísticas, reorganização de tabelas e índices e *backup*;
6. Falta de recursos para evolução de esquemas de forma *on-line* - Os SGBD devem permitir a mudança de aspectos do esquema sem a necessidade de tirar o banco do ar. As aplicações não são estáticas. Tabelas são modificadas, colunas são apagadas, tipos de dados modificados, índices renomeados e todas essas alterações devem ser feitas com o banco *on-line*;
7. Falta de padrões de interface com outros sistemas - SGBD atuais não possuem capacidade adequada de integração harmoniosa com outros sistemas. Normas devem ser desenvolvidas para facilitar a comunicação e o compartilhamento de informações; e
8. Falta de características para redistribuição de carga de trabalho - A maior parte dos SGBD atuais não trata a variação da carga de trabalho ao longo do tempo. SGBD Autônomos devem adaptar-se em função da intensidade de trabalho e tendências.

Apesar de todas as dificuldades técnicas e de integração, percebemos que o caminho para o Sistema Autônomo de Gerenciamento de Banco de Dados (SAGBD) começa a ser trilhado e que as próximas versões dos produtos, cada vez mais, serão dotadas de características de autonomia, o que mudará radicalmente o cenário de gerenciamento de bancos de dados e da atuação dos profissionais que os utilizam. O surgimento de um SGBD considerado totalmente autônomo ainda necessitará de tempo e bastante esforço de pesquisa. Dentre as vertentes de pesquisa que buscam a concretização do SAGBD, existe uma que desenvolve características de autonomia para o gerenciamento através do uso de agentes inteligentes de *software*.

O trabalho a ser desenvolvido seguirá a vertente citada acima, buscando implementar uma solução que minimize o problema da falta de recursos para evolução de esquemas de forma *on-line* (**item 6** da lista descrita anteriormente), que segundo Elnaffar et al. (2003) consomem muita força de trabalho dos DBA e ainda não estão presentes da forma que deveriam nos SGBD.

A solução a ser construída baseia-se em conceitos da Computação Autônoma aplicados à propagação de modificações/atualizações para realizar evolução de esquemas

em ambientes de bancos de dados distribuídos e heterogêneos. Isso será feito utilizando agentes inteligentes, que vão compor uma camada/módulo exterior ao SGBD e através dela realizar as operações necessárias para atender às requisições dos DBA. De forma pró-ativa, procurará evitar falhas humanas e repetição de trabalho. A solução atenderá as mudanças ocorridas no negócio, refletindo a nova realidade nos diversos esquemas do ambiente de banco de dados de forma automática, bastando o DBA informar alguns dados de entrada.

No Capítulo 3 falaremos sobre evolução de esquemas, técnicas de propagação de atualização e até qual nível elas podem ser automatizadas com o uso de Computação Autônoma no contexto de SGBD.

CAPÍTULO 3

# Evolução de Esquemas e Propagação de Atualizações

Este capítulo descreve os principais conceitos sobre Evolução de Esquemas, apresentando uma contextualização e necessidade do seu uso, bem como traz informações sobre como é desenvolvido um projeto de Sistema de Banco de Dados. Entender como é feito o projeto, suas fases e artefatos de saída é essencial para realizar uma evolução de esquemas com qualidade. Também são apresentados os requisitos básicos a serem atendidos para uma evolução de esquemas satisfatória e uma descrição das principais técnicas e estratégias utilizadas. O capítulo é finalizado com a análise crítica de alguns trabalhos relacionados, considerações e complementações sobre o que foi apresentado.

## 3.1 Introdução

Devido ao rápido crescimento da globalização mundial e o conseqüente surgimento de novos mercados, serviços e oportunidades, as empresas se vêem forçadas não só a procurarem novos clientes, mas também a manterem os já existentes. Dessa forma, elas têm mais chances de se tornarem promissoras num ambiente tão dinâmico e competitivo. Para que isso seja possível, a procura por melhoras na qualidade de seus produtos ou serviços oferecidos é constante. Esse dinamismo dos negócios empresariais faz com que os sistemas de informação que os apoiam precisem evoluir rapidamente para se adaptarem às novas realidades. Por causa disso, a capacidade de adaptação desses sistemas aliada à qualidade com que essas adaptações são incorporadas e postas em produção vem sendo considerada um fator determinante para a competitividade empresarial.

Um dos maiores desafios de se construir e manter aplicações de grande porte com ciclos de vida longos é lidar com as inevitáveis mudanças de requisitos que ocorrem

com o tempo. Muitas dessas aplicações dependem de sistemas de bancos de dados que na maioria das vezes, sofrem reflexos diretos nos seus esquemas devido às alterações na realidade que modelam. Existem vários motivos pelos quais alterações ocorrem. Primeiro, os especificadores não sabem antecipadamente, ou não são capazes de exprimir, todas as funcionalidades de uma aplicação de grande porte ou complexa. Só a utilização do sistema e a obtenção de experiência permitirão a devida formulação dos requisitos. Segundo, o ambiente em que a aplicação está inserida está em constante mudança. Uma aplicação viável deve ser capaz de acomodar mudanças. Conseqüentemente, alterações (evoluções) no esquema de banco de dados são necessárias para garantir que o sistema reflita os requisitos com a maior precisão possível em todos os momentos [Sjoberg, 1993].

Quando um sistema computacional sofre uma manutenção evolutiva ou adaptativa para se adequar às novas realidades, essa manutenção geralmente não se restringe apenas na alteração de seu código fonte, pois muitas vezes ela tem impactos no banco de dados que lhe dá suporte. Quando isso ocorre, as alterações devem ser refletidas nos respectivos esquemas de banco de dados. Isso é de essencial importância para que seja possível manter a consistência dos dados armazenados com a realidade que representam. Esquemas de banco de dados, dependendo dos domínios das aplicações, costumam sofrer muitas alterações ao longo do seu ciclo de vida. Estudos apontam um aumento médio de 139% no número de relacionamentos e de 274% no número dos atributos em relação à versão inicial do esquema [Blaschka, 2000].

O controle consistente e inteligente das mudanças estruturais, das instâncias de dados e das aplicações que se utilizam destes dados se faz fundamental, pois uma simples mudança mal gerenciada pode acarretar em sérias conseqüências para o sistema como um todo e trazer grandes impactos aos negócios empresariais. Apesar do tema Evolução de Esquemas de Banco de Dados já ser há bastante tempo discutido, com vários trabalhos na literatura, não encontramos uma norma ou padrão que o formalize como um processo de trabalho a ser seguido. Essa característica possibilita sua abrangência, generalidade e desdobramento para inúmeras situações, o que dá margem a variações entre as diversas pesquisas existentes sobre este assunto, permitindo tratá-lo sobre diversos pontos de vista.

É nesse contexto que vamos explanar alguns tópicos sobre Evolução de Esquemas.

## 3.2 Questões sobre Evolução de Esquemas

Nesta seção, apresentaremos os conceitos básicos sobre Evolução de Esquemas e o porquê de sua utilização. Faremos isso respondendo a duas questões que são essenciais para o entendimento do tema: o que é e para que serve evolução de esquemas.

### 3.2.1 O que é Evolução de Esquemas?

Segundo Camolesi e Traina (1996) a manutenção de um sistema de banco de dados não envolve somente a troca ou reciclagem de componentes de *software*, como: ambientes, aplicativos, SGBD e estruturas do banco de dados, mas também componentes de *hardware* (computadores e equipamentos), *peopleware* (usuários comuns, programadores e DBA) e até mesmo técnicas utilizadas para sua modelagem e implementação. Partindo para um escopo mais específico de componentes de banco de dados (esquemas de dados e instâncias), a flexibilidade para a realização de manutenções no banco é determinada pela maneira como foi projetado (adequação do projeto) e pela capacidade funcional de seus respectivos SGBD. É essa habilidade de flexibilidade para incorporar mudanças que Camolesi denomina *evolução*. Ela pode depender tanto do modelo de dados quanto de sua implementação, variando de caso para caso com uma maior ou menor flexibilidade.

Na literatura, podemos encontrar outras definições para evolução de esquemas, como por exemplo: evolução de esquemas é o processo de manter as mudanças esquemáticas de um repositório de dados [Lakshmanan, 1993]. Para Roddick (1992), evolução de esquemas é a habilidade do sistema de banco de dados responder às mudanças do mundo real, permitindo o esquema evoluir conforme essas mudanças. Para Rahm e Bernstein (2006), evolução de esquemas é a capacidade de mudar esquemas de dados implantados. Como podemos ver, existem várias definições para evolução de esquemas, mas todas exprimem a idéia principal de refletir mudanças da realidade modelada nos esquemas de dados com o objetivo de manter a consistência.

### 3.2.2 Causas da Evolução de Esquemas

Sistemas de banco de dados exigem a capacidade de lidar com um grande volume de dados persistentes, buscando sempre garantir a consistência deles. Toda essa informação armazenada, geralmente, é usada por longos períodos de tempo pelas aplicações e é freqüentemente atualizada, principalmente os esquemas e instâncias de dados. Esquemas utilizados em um determinado domínio tendem a evoluir com o passar do tempo, inclusive em repositórios onde já existem dados válidos. Diversos fatores podem levar à necessidade dessa evolução. Adiante veremos os principais.

De uma forma geral, a manutenção do banco de dados com relação ao esquema tinha como foco principal a diminuição do desempenho causado pelas muitas e constantes atualizações (restruturações) do esquema em uso. Quando não devidamente tratada, esta queda no desempenho elevava os custos de acesso às informações (ver Figura 3.1), aumentando o tempo de resposta das transações a um patamar inviável de utilização do banco de dados (ou base de dados) [Sockut & Goldberg, 1979].

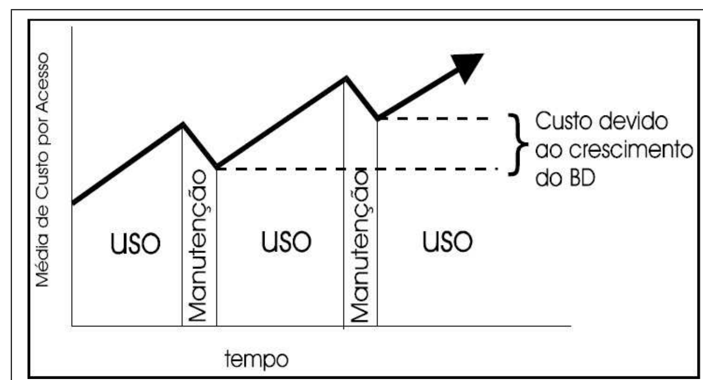



**Figura 3.1** Manutenção de esquemas com base na diminuição do desempenho do banco de dados. Fonte: adaptado de [Sockut & Goldberg, 1979].

Atualmente, realizar evolução de esquemas é uma tarefa bem mais complexa. Existem diversas questões a serem consideradas além do desempenho, tais como: o que deve ser alterado? Qual o impacto da alteração? Quando deve ser realizado? Como realizar? Quais os benefícios? Qual o custo desse processo? Quem será afetado? e outras, que devem compor a política de manutenção de um sistema de banco de dados. Mesmo com tantas questões a serem analisadas e com um nível de complexidade alto, o elevado grau de aplicabilidade deste tema o faz funcionalmente necessário em várias situações, por

diferentes motivos, dentre os quais podemos destacar [Camolesi & Traina, 1996]:

1. Correção do esquema de dados diante de problemas encontrados durante a fase de operação com o banco de dados (falta de testes ou falhas humanas);
2. Adaptação do esquema de dados a novos componentes ou circunstâncias que envolvem o sistema de banco de dados e seu ambiente de utilização (interfaces ou processos);
3. Refinamento de componentes do esquema que foram pouco detalhados durante a fase de projeto e, portanto, estavam semanticamente pobres (especificação deficiente);
4. Atualização de componentes do esquema de modo a acompanharem as constantes modificações de requisitos dos usuários e as evoluções do empreendimento;
5. Experimentação de novas concepções do esquema de dados em momentos quando buscam-se alternativas para a modelagem do banco de dados; e
6. Incorporação de novos elementos ao esquema de dados.

Segundo Rahm e Bernstein (2006), a necessidade de evolução de esquemas ocorre muito mais freqüentemente devido a casos de mudanças de requisitos e migração de plataforma. Baseado nisso, os itens 2, 3 e 4 receberam uma maior atenção e serão comentados adiante. Esses três itens também fazem parte das principais motivações para o desenvolvimento da solução proposta neste trabalho e foram levados em consideração na sua construção. Focando neles, o custo de uma mudança num sistema de banco de dados devido a alterações ou detalhamento de requisitos está intimamente ligado ao impacto dessa alteração no sistema como um todo, à qualidade do seu projeto e nível em que se encontra. Por exemplo, se tivermos um sistema de banco de dados em desenvolvimento que esteja no nível lógico do projeto, e que tenha passado por um nível conceitual bem documentado e especificado, certamente lidar com uma mudança de requisitos será menos custoso que num sistema já concluído e com um projeto deficiente.

Evolução de esquemas em sistemas de banco de dados traduz mudanças de requisitos referentes à realidade que modelam, sejam funcionais, organizacionais ou de desempenho. Modificações ou detalhamento de requisitos funcionais e organizacionais geralmente são considerados os mais críticos e custosos de se implementar. Eles são mais comuns em organizações que possuem negócios dinâmicos, que mudam suas regras freqüentemente. Modificações contínuas no esquema são necessárias para garantir

que o sistema reflita os requisitos com a maior precisão possível.

Em um sistema de banco de dados implementado e em produção, quando uma alteração de requisitos é feita, ou algum novo requisito é criado, ela é analisada, detalhada e as mudanças resultantes são propagadas em todos os níveis do projeto. No caso de requisitos de desempenho, muitas vezes eles não afetam o projeto conceitual e são implementados direto no esquema físico. À medida que essas mudanças vão sendo propagadas nos níveis do projeto, as referidas documentações devem ser atualizadas com as novas considerações. Quanto mais cedo as mudanças de requisitos forem identificadas, melhor, pois elas não precisarão ser propagadas em todos os níveis do projeto (conceitual, lógico e físico), diminuindo custos.

Sabemos que definir perfeitamente todos os requisitos nas fases iniciais do projeto do banco de dados não é uma tarefa fácil. Requisitos mudam com o tempo e um sistema bem projetado e documentado deve ser capaz de suportar mudanças, permitindo a propagação delas inclusive nos esquemas e instâncias de dados, se necessário. Essa propagação nas instâncias de dados deve ser feita de forma muito criteriosa, visto que um erro pode inserir inconsistências no banco, causando problemas seríssimos para as aplicações que fazem uso desses dados. Esse problema pode ser agravado em ambientes com várias aplicações integradas, dependentes dos mesmos dados. Tratar a evolução de esquemas é uma tarefa necessária não somente durante a fase de desenvolvimento de um banco de dados, mas também durante toda sua fase operacional. Sjoberg (1993) comprova esse fato: mudanças no esquema são tão comuns na fase de desenvolvimento de um banco de dados quanto em sua fase operacional.

O principal objetivo para o suporte à evolução de esquemas em sistemas de banco de dados é preservar a integridade dos dados. Se o novo esquema afetará a visão dos dados antigos, se as consultas baseadas no esquema antigo continuarão a rodar com os dados novos, se os dados antigos poderão ser visualizados no esquema novo, entre outros. Mudanças causarão impacto e precisam ser estudadas para verificar como serão propagadas da melhor forma para evitar inconsistências [Saccol & Edelweiss, 2005].

Um suporte efetivo à evolução de esquemas é um desafio, levando em conta que as mudanças têm que ser propagadas corretamente e de maneira eficiente para as instâncias de dados, *views* e outros. Idealmente, lidar com essas mudanças deve exigir o mínimo de intervenção humana e indisponibilidade de sistema [Rahm & Bernstein, 2006].

### 3.2.3 Modificação, Evolução e Versionamento de Esquemas

Os conceitos de modificação, evolução e versionamento de esquemas são comumente confundidos pelos estudiosos da área de banco de dados. Isso ocorre porque a diferença entre eles não é muito clara na literatura atual e muitas vezes são utilizados como sinônimos [Roddick, 1995] quando na verdade não são. No entanto, o trabalho de Jensen [Jensen et al., 1998] traz um glossário que apresenta um consenso entre as definições desses três conceitos, a saber:

- **Modificação de esquema** - O sistema de banco de dados permite modificações na definição do esquema de um banco de dados populado, mas não se preocupa em manter a consistências dos dados;

- **Evolução de esquema** - O sistema de banco de dados permite modificações na definição do esquema, sem perda de dados existentes, preocupando-se com a consistência; e

- **Versionamento de esquemas** - O sistema de banco de dados permite acessar todos os dados, antigos e novos, através de versões definidas, que são guardadas a cada nova alteração do esquema.

Roddick (1995) argumenta que evolução de esquemas não implica necessariamente num suporte completo para guardar o histórico de evolução por parte do sistema de banco de dados, mas somente a habilidade para alterar as definições do esquema conceitual sem a perda de informações. A preocupação da evolução de esquemas em manter a consistência dos dados pode ser vista nas suas várias estratégias de propagação de mudanças para as instâncias de dados, as quais veremos com mais detalhes na Seção 3.5.2. Já o versionamento de esquemas, *que é uma abordagem da evolução de esquemas*, requer que o histórico das mudanças seja mantido para possibilitar a manutenção de definições passadas do esquema. Versionamento de esquemas identifica pontos estáveis na definição e rotula esta definição para futuras referências. Evolução de esquemas não requer a capacidade de versionar dados, nem requer que o banco de dados forneça um mecanismo de visualização para definições passadas de esquema.

A modificação de um esquema não gera necessariamente uma nova versão. Modificações de esquema tendem a ser de granularidade menor que as versões definidas e não

garantem a consistência dos dados. Versões tendem a ser rotuladas ou pela data da transação que alterou o esquema ou por algum método definido pelo usuário; já mudanças de evolução de esquemas são geralmente referenciadas pela data da modificação.

## 3.3 Projeto de um Sistema de Banco de Dados

Para um melhor entendimento da propagação das mudanças de requisitos nos esquemas é importante compreender como está estruturado um projeto de sistemas de banco de dados, ou simplesmente projeto de banco de dados. Adiante, explicaremos seus níveis e mostraremos a importância de ser elaborado com qualidade.

O projeto de um sistema de banco de dados é um processo longo e meticuloso, requer domínio do que se quer modelar, experiência e conhecimento técnico. Ele é realizado através de processos elementares, baseados em três níveis de abstração: conceitual (representa requisitos do domínio), lógico (ênfase na eficiência de armazenamento) e físico (ênfase na eficiência de acesso). Geralmente para efetuar o projeto, consideramos três tipos de requisitos: funcionais (requisitos em termos de funções do sistema), organizacionais (refletem aspectos da organização) e técnicos. A Figura 3.2 representa estes três níveis de abstração descritos.

É importante frisar que antes do projeto atingir o nível conceitual, uma análise prévia dos requisitos deve ter sido feita. Esperamos que especificações de requisitos estejam definidas e documentadas através de discussões sobre o domínio a ser modelado.

No fluxo de construção de um sistema de banco de dados, começamos com o estudo e análise do ambiente que será modelado (mini-mundo). É nesse momento que são identificados e especificados todos os tipos de requisitos desejados, sejam funcionais, organizacionais ou técnicos, lembrando que essa especificação delimita o escopo do sistema de informação. Uma vez especificados, os requisitos são modelados no nível de mais alto grau de abstração, o conceitual. Essa modelagem dá origem ao esquema conceitual, que contempla os requisitos funcionais e organizacionais numa visão mais ampla. Eles vão sendo refinados e cada vez mais detalhados nos outros dois níveis, dando origem respectivamente aos esquemas lógico e físico (ver Figura 3.2). O esquema lógico representa precisamente as estruturas de dados e relacionamentos, enquanto o físico é responsável por atender às restrições de desempenho. Por exemplo: é no nível

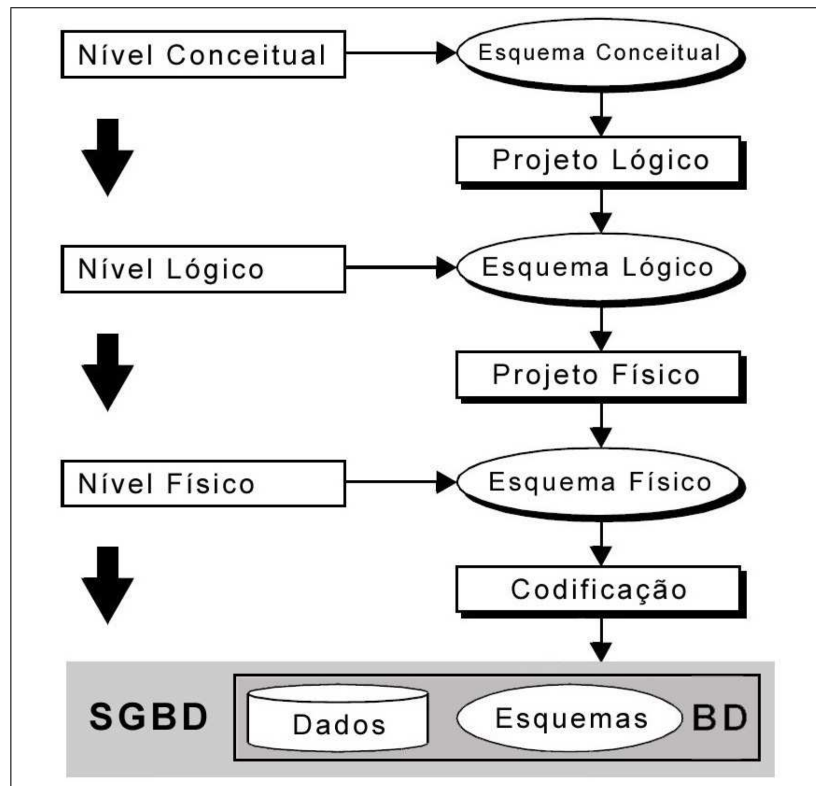



**Figura 3.2** Níveis de abstração do projeto de BD. Fonte: adaptado de [Hick & Hainaut, 2002].

físico que buscamos soluções para atender requisitos como tempo de resposta pequeno, através da criação de um índice em determinada tabela. Do esquema físico, parte a implementação do código fonte (SQL) que é aplicado no SGBD alvo, para a efetiva criação do banco. Por esse motivo, existe uma dependência entre o esquema físico e o SGBD. Os dois esquemas, lógico e físico, são responsáveis por contemplar os requisitos técnicos [Wattiau et al., 2002].

A facilidade de manutenção e eficiência de um sistema de banco de dados está intimamente ligada à qualidade de seu projeto. Para que isso seja atingido de forma satisfatória, é esperado que os níveis conceitual, lógico e físico sejam seguidos seqüencialmente, do mais alto grau de abstração para o menor, pois eles ajudam a desenvolver estruturas de dados com qualidade, o que é imprescindível para garantir a legitimidade do banco de dados, facilitando a manutenção do sistema de aplicação. As construções resultantes de todos os níveis devem ser devidamente documentadas e estarem de acordo com os requisitos identificados.

Sabemos que considerar os sistemas de banco de dados, tendo suas especificações para cada nível de abstração do projeto realizadas, completas e documentadas é no mínimo irreal em muitas situações. Existem muitos sistemas legados que foram construídos sem ao menos terem um projeto de desenvolvimento. Nestes, o código fonte e os *scripts DDL* [1] são os únicos artefatos disponíveis.

Sistemas com documentação incompleta ou projeto deficiente também são comuns. Para casos como esses, recorre-se a técnicas de *engenharia reversa* [2]. Quando um projeto de sistema de banco de dados é feito desde o início, é extremamente recomendado modelar a realidade a que ele se refere da maneira mais fidedigna possível para minimizar impactos de futuras mudanças de requisitos ou especificação incompleta/falha. Um projeto deficiente dificulta a manutenção/evolução do sistema de banco de dados, mais especificamente dos esquemas de banco de dados.

## 3.4 Requisitos para Evolução de Esquemas

Focando na necessidade de fornecer uma maior capacidade de manutenção aos esquemas nos sistemas de banco de dados, a maior parte dos trabalhos sobre evolução de esquemas existentes na literatura podem ser classificados em três linhas principais de pesquisa [Camolesi & Traina, 1996]:

- **Projeto** - Envolve o conceito de modelo de dados, ou seja, os conceitos sobre esquemas e a metodologia utilizada para produção de tais esquemas;
- **Forma** - Relacionado ao problema de evolução de esquemas físico e suas conseqüências de reestruturação e reformatação no banco de dados; e
- **Lógica** - Relacionado ao problema de evolução física e conceitual do esquema externo (as visões dos usuários). Refere-se, especificamente, às operações de alteração do esquema lógico, às técnicas de evolução empregadas e às estratégias de propagação das atualizações nas instâncias de dados.

[1] *Data Definition Language (DDL)* é uma linguagem para definição de estrutura de dados.

[2] Metodologias para o reconhecimento e representação dos aspectos estruturais, funcionais e comportamentais de um sistema já implantado [Chikovsky & Cross, 1990].

Ainda no contexto de fornecer uma maior capacidade de manutenção e com o objetivo de preservar a consistência do banco de dados, a complexidade da evolução de esquemas pode ser classificada em cinco níveis, são eles:

- **Representação de esquema** - Identificação dos aspectos do esquema que devem sofrer modificações;
- **Operações de modificação de esquema** - Identificação das operações de modificação permitidas para o esquema;
- **Consistência na modificação de esquema** - Maneira como as modificações do esquema são qualificadas e propagadas no banco de dados;
- **Integridade do banco de dados** - Identificação da forma como as instâncias vigentes no banco de dados são afetadas frente à modificação do esquema; e
- **Disponibilidade do banco de dados** - Capacidade do banco de dados permitir a execução das aplicações normalmente após a modificação do esquema.

Cada modelo de dados define elementos específicos em seu esquema, ou seja, um conjunto de operações para promover a evolução e restrições de integridade para a execução das ações evolutivas que geram um novo esquema. Toda ação de alteração de esquema deve passar por filtros de integridade, nos quais a consistência do novo esquema deve ser verificada através de um conjunto de restrições para cada uma das alterações permitidas [de Matos Galante, 2003].

O gerenciamento da evolução de esquemas durante a fase operacional é um problema complexo, pois cada mudança deve levar em consideração as dependências e instâncias previamente armazenadas. Mesmo que não se leve em conta o histórico da evolução do esquema com o passar do tempo, cada mudança deve levar em consideração replicações de esquemas e elementos que sofrem impacto direto, como por exemplo: *triggers*, *procedures* e *packages* de banco relacionadas ao esquema físico modificado.

A inclusão de maiores facilidades para evolução de esquemas envolve a solução de dois problemas fundamentais: o primeiro é a semântica de modificação de esquemas, que se refere aos efeitos das modificações no próprio esquema; o segundo é a semântica de propagação de mudanças, que descreve a forma como essas modificações nos esquemas afetam as instâncias do banco de dados. O primeiro problema envolve a verificação

e manutenção da consistência após as modificações no esquema, ao passo que o segundo envolve a consistência dos dados vigentes de acordo com as mudanças realizadas.

No nível lógico ou físico, as alterações no banco de dados em decorrência de uma evolução de esquemas, levando em conta o grau de complexidade e abrangência, podem ser realizadas através de duas estratégias básicas, são elas [Camolesi & Traina, 1996]:

- ***Off-line*** - Esta estratégia interrompe todos os processos em andamento enquanto o banco de dados está sendo atualizado. É executada em um período de pouca requisição de dados, pois os usuários comuns têm seus pedidos de acesso negados durante o período em que as alterações estão sendo realizadas; e

- ***On-line*** - Esta estratégia mantém o sistema de banco de dados em operação enquanto se realiza a evolução dos esquemas. É recomendada para aqueles que são críticos para seus usuários ou para bancos que envolvem grande volume de dados. A reorganização e utilização do banco são processos concorrentes, controlados de forma que o usuário comum possa ter suas requisições atendidas enquanto uma porção desse banco é atualizado. A Figura 3.3 ilustra as duas estratégias.

## 3.5 Evolução dos Esquemas Conceitual e Lógico

Na maioria das vezes, a evolução dos esquemas conceitual e lógico ocorre de forma interligada, ou seja, uma evolução no esquema conceitual geralmente acarreta numa evolução/modificação do esquema lógico. Modificações no esquema lógico podem implicar tanto em alterações das instâncias de dados envolvidas como em modificações nos programas aplicativos que utilizam estas instâncias. Vale a pena frisar, que a alteração / evolução das instâncias de dados num determinado banco não é necessariamente causada pela evolução dos esquemas correspondentes. Isso quer dizer que mesmo não ocorrendo modificações no esquema de dados, pode existir a necessidade de modificarmos as instâncias por outros motivos, como por exemplo: uma funcionalidade implementada de forma errada pode inserir dados no banco que não condizem com a realidade; com a correção dessa funcionalidade, as instâncias correspondentes também devem ser corrigidas.

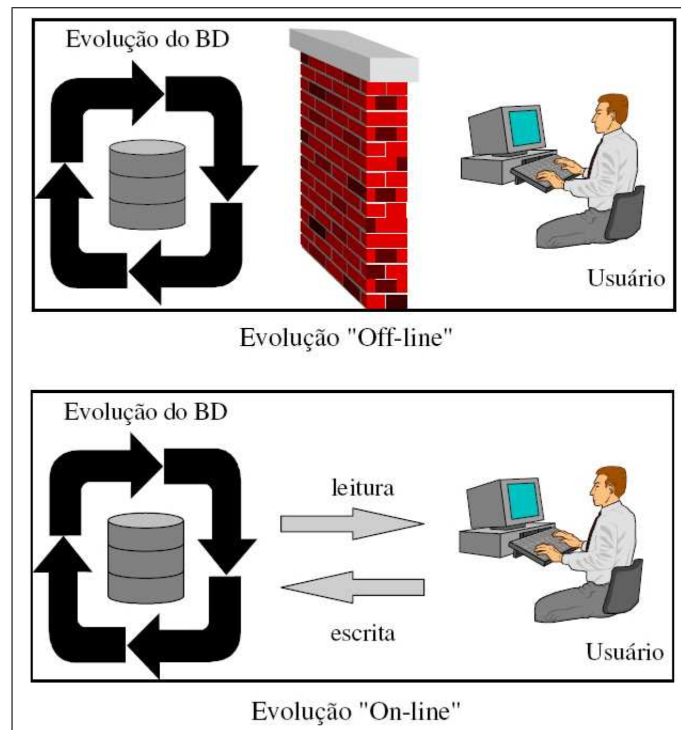



**Figura 3.3** Estratégia *off-line*, quando os serviços do banco de dados são parados para a realização de atualizações, e a estratégia *on-line*, quando os serviços continuam no ar. Fonte: adaptado de [Camolesi & Traina, 1996].

Devido à estreita relação entre instância de dados e esquema, surge a necessidade de técnicas de evolução de esquema e de estratégias de propagação nas instâncias de dados, que ajudam a minimizar os impactos negativos da evolução e evitam inconsistências no banco de dados.

### 3.5.1 Técnicas para tratar a Evolução de Esquemas

Na literatura, existe uma variedade de trabalhos com técnicas propostas que ajudam a efetuar mudanças de forma segura, evitando a degradação do banco de dados pela inserção de erros. Porém, não foi encontrado nada que classifique formalmente essas técnicas, por isso elas não têm denominações comuns e não há especificações que digam quais são as melhores ou piores. Dessa forma, não é razoável dizer qual a melhor ou pior técnica, pois todas têm capacidade de serem utilizadas nos mais diferentes

contextos das aplicações. Cada uma delas foi aprimorada ou estendida tentando atingir uma compreensão genérica dos problemas. O que deve acontecer é a escolha da técnica mais apropriada para determinado caso, que deve ser bem analisado para que possamos atingir os objetivos esperados com o menor custo.

O trabalho de Andany et al. (1991) traz três abordagens principais para tratar a evolução de esquemas. Vejamos em detalhes cada uma delas.

- **Criação/Modificação de Esquemas (*Creating/Modification)*** - Esta técnica é baseada na alteração do esquema de dados corrente. Para que seja aplicada é criado um novo esquema, que pode ser derivado ou não de uma cópia do esquema implantado. As modificações no novo esquema devem ser preferencialmente de pequeno porte, ou seja, poucas e pequenas alterações em relação ao esquema implantado, mesmo que tenham grande influência sobre as instâncias de dados. Na maioria dos casos, as modificações do esquema englobam operações simples, tais como: criação, deleção e modificação, que são executadas através de comandos nativos contidos na *Data Definition Language (DDL)*. Geralmente o esquema antigo e seus dados correspondentes são substituídos pelo novo esquema e seus novos dados. Isso pode acarretar perda de informação, por exemplo, alguns dados existentes podem já não serem relevantes no novo esquema e, portanto, podem ser descartados. Pode também acarretar invalidez dos aplicativos que usavam o esquema antigo de dados, tornando-os incapazes de operar com o novo esquema [Banerjee et al., 1987b]. Nada impede que o esquema antigo continue armazenado para efeito de revisões e auditoria, porém isso vai depender do modelo de dados utilizado e do suporte a instâncias de dados antigas. Todos esses fatores tornam essa abordagem inadequada na maioria dos casos reais atuais, apesar de ela ser popular entre os SGBD existentes;

- **Suplementação de Esquemas (*Additional*)** - Esta técnica consiste em adicionar novos elementos ao esquema implantado e, assim como a técnica de Criação / Modificação, é baseada na alteração do esquema de dados corrente. A diferença básica é que o uso da técnica de suplementação é mais comum quando ocorrem mudanças na realidade modelada que acarretam numa grande quantidade de novos componentes que precisam ser integrados ao esquema atual. Esta técnica oferece meios para a adição de sub-esquemas a estruturas já existentes, além de

permitir a incorporação gradativa do esquema de dados em função de instâncias já armazenadas, como exemplo temos as derivações *botton-up* [Zdonik, 1993] e *incomplete information* [Zicari, 1990]. A inconsistência dos dados é um fator crítico que precisa ser tratado com o uso desta técnica e, para isso, devem existir controles específicos que permitam a adição de novos elementos ao esquema sem comprometer de forma alguma as instâncias; e

- **Versionamento de Esquemas (*Schema Versioning*)** - Esta técnica propõe a criação de versões a partir de um esquema base como forma de realizar sua evolução. O esquema antigo e seus dados correspondentes são preservados e continuam a ser utilizados por aplicações existentes, porém uma nova versão do esquema é criada incorporando as alterações desejadas [Kim & Chou, 1998]. Dependendo do tamanho do banco de dados, versionamento de esquemas pode ser um processo bastante longo e complexo. Para minimizar essa dificuldade, podemos recorrer ao desenvolvimento em paralelo de versões candidatas. Concluído o processo de criação dessas possíveis versões, elas são comparadas, sendo escolhida a que melhor representa a realidade modelada em sua nova concepção, promovida a uma versão do esquema. Em muitos casos, adotar a abordagem por versionamento traz muitos problemas de desempenho, pois ela requer uma grande quantidade de recursos de *hardware* e *software*, tais como: memória, processador, espaço em disco, aplicações para realizar as funções de conversão [3] necessárias para atender às requisições dos usuários, dentre outras. Todo esse consumo de recursos é necessário para que o correto gerenciamento e manutenção de todas as versões acumuladas do esquema, com suas respectivas operações e dados, sejam possíveis. Isso gera um custo muito alto e a longo prazo torna-se difícil manter o banco de dados devido ao aumento do seu tamanho e do número de versões dos esquemas.

Muitos trabalhos de pesquisa têm concentrado seus esforços na evolução de esquemas conceituais, independente de qualquer influência do modelo de dados lógico. Eles têm focado mais em questões como: estabilidade, qualidade, mudanças e semântica de

[3] Funções de conversão garantem que as instâncias geradas em qualquer versão de esquema permaneçam visíveis e atualizáveis sob qualquer perspectiva de versão, através de implementações que realizam uma adaptação, restaurando os dados de uma versão para outra [de Matos Galante, 2003].

esquemas [Gomez & Olive, 2002; Gomez & Olive, 2003]. Um exemplo disso é o caso do trabalho desenvolvido por Wedemeijer [Wedemeijer, 2000]. Ele propõe um *framework* definido por um conjunto de métricas que controlam a estabilidade dos esquemas conceituais depois do processo de evolução.

Considerando outras linhas de pesquisa, há também o caso da evolução de esquemas baseado em ontologias. Ela trata os mesmos problemas da evolução de esquemas de banco de dados, embora tenha suas peculiaridades, tais como: vocabulários controlados, taxonomias e representação de conhecimento baseado em regras. Pesquisadores da área de banco de dados distinguem evolução de esquemas e versionamento de esquemas, enquanto para ontologias não há distinção entre esses dois conceitos. Contudo, eles são usados para entender o problema da evolução de banco de dados [Lammari et al., 2003] e podem ser utilizados em diversos contextos. A ontologia pode ser vista como uma rede semântica que classifica a evolução ou a mudança em uma ou mais categorias dependendo da sua importância, semântica, informação, dependência de aplicação e assim por diante [Bounif, 2004].

### 3.5.2 Estratégias de Propagação nas instâncias

Independente da técnica utilizada, uma vez que as modificações são aplicadas no esquema de dados, elas devem ser propagadas para instâncias a fim de manter a consistência. Em alguns casos pode ocorrer das modificações no esquema não causarem impactos nas instâncias como, por exemplo: a criação de um índice em uma determinada tabela para aumentar o desempenho. Diferente das técnicas de evolução de esquemas, podemos encontrar na literatura classificações para as estratégias de propagação nas instâncias de dados. O trabalho de Bjornerstedt (1989) é um exemplo disso e descreve de maneira bem elementar uma abordagem usada para repassar o esquema de dados com as alterações aplicadas para as instâncias de dados.

Como as estratégias de propagação não são estáticas, é possível adaptá-las a domínios específicos em que são utilizadas. Por esse motivo, podemos encontrar muitas variações, que atualmente são enquadradas em cinco categorias que veremos a seguir [Camolesi & Traina, 1996].

- ***Copying*** - Nesta estratégia, após a concretização do novo esquema, é realizada a transposição das instâncias relacionadas à evolução, através de cópias dos dados do esquema antigo para o novo esquema (nova situação). Essas cópias podem ser feitas de uma única vez (transposição imediata) ou em subconjuntos das instâncias (transposição incremental) até que todas sejam copiadas. Essa estratégia mostra-se como uma abordagem muito simples e muito utilizada nos SGBD, porém consome muito tempo para finalizar as cópias do antigo para o novo esquema. Esse problema agrava-se ainda mais quando se trata de bancos com um grande volume de dados, o que exige implementações cada vez mais otimizadas por parte dos DBA;

- ***Updating*** - Nesta estratégia, as instâncias de dados relacionadas à evolução não são copiadas, mas sim convertidas para ficarem de acordo com o novo esquema. Essa conversão pode ser realizada logo após a criação do novo esquema (conversão imediata) ou durante a manipulação dos dados (conversão incremental). No caso da conversão incremental, à medida que o SGBD vai acessando as informações, é verificada a existência de alguma indicação de evolução para o esquema atual. No caso positivo, o SGBD realiza a atualização dos dados acessados para a nova situação. Assim como na estratégia de *Copying*, a conversão imediata consome muito tempo para concretizar todas as atualizações de uma única vez para o esquema novo. Já a conversão incremental apresenta lentidão nas requisições que manipulam dados ainda não convertidos, pois o SGBD deve atualizar os dados acessados para o esquema novo antes de retornar algo, o que causa um aumento no tempo de resposta da transação;

- ***Screening*** - Esta estratégia, também conhecida como mapeamento, prorroga indefinidamente a atualização das instâncias de dados, mesmo após o novo esquema ser criado. Dessa forma, nenhuma alteração real é feita nos dados, até que ela seja exigida por um usuário que tenha permissão para tal. Análogo à estratégia de *Updating* na abordagem de conversão incremental, quando a informação é acessada pelo SGBD, é verificado se existe algum tipo de indicação de evolução. Se existir, o SGBD utiliza um artifício de adaptação lógica da informação acessada para adaptá-la à nova situação do esquema, que emula instâncias modificadas. Vale

ressaltar que nesse processo de emulação de instâncias modificadas não existe nenhum tipo de atualização física no banco. O fato de criar uma ilusão de instâncias de dados modificadas compromete a velocidade das operações que manipulam dados por parte do SGBD, pois todas as informações retornadas ao usuário devem estar transformadas/emuladas de acordo com o novo esquema;

- ***Versioning*** - Nesta estratégia, cada novo esquema de dados instanciado leva à criação de versões das instâncias de dados relacionadas à ele. Essas versões podem ser criadas gradativamente pelo usuário, desde que devidamente gerenciadas pelo SGBD, ou podem ser criadas automaticamente através de cópias ou conversões de uma versão de instância para outra. Cada versão das instâncias também pode apresentar diferentes graus ou igualdade de flexibilidade no que se refere a sua manipulação pelo SGBD. Isso exige um maior grau de controle das instâncias, que geralmente é realizado por um sistema especializado de Controle de Versões. Com um sistema dessa natureza, é possível resolver conflitos e gerenciar de forma satisfatória atividades de criação, manipulação e consulta de várias versões das instâncias, que podem estar armazenadas na mesma base de dados ao mesmo tempo; e

- ***Materializing*** - Essa estratégia é bem diferente de todas apresentadas anteriormente, pois não objetiva atingir as instâncias de dados já existentes através de operações de cópia, atualização, conversão lógica ou criação de versões. Ela cria novas instâncias de acordo com o novo esquema de dados (materialização). O que já existe, instâncias e esquemas antigos, permanece inalterado e pode ser utilizado a qualquer momento. O ponto positivo dessa estratégia é que ela não requer controles específicos e não traz nenhum empecilho ao processo de manipulação das instâncias, mas exige do SGBD a capacidade de reconhecer o esquema de dados novos de cada uma das instâncias armazenadas e ter a flexibilidade para materializá-las em momentos que o usuário precisar utilizá-las.

Alguns SGBD fazem uso da conversão imediata para propagar atualizações nas instâncias, como é o caso do *GEMstone* [4] [Butterworth et al., 1991], *ObjectStore* [5] e *Ora-*

[4] http://www.gemstone.com
[5] http://www.progress.com/objectstore

*cle* [6]. Já o *Orion* [Banerjee et al., 1987a] utiliza-se da conversão incremental e o *Encore* [Zdonik & Mitchell, 1991] da emulação de instâncias modificadas. Ainda existem SGBD que combinam as abordagens de conversão imediata e incremental, como é o caso do *Sherpa* [Nguyen & Rieu, 1989] e O2 [Ferrandina et al., 1995].

### 3.5.3 Aspectos sobre o uso das Técnicas de Evolução e Estratégias de Propagação

Todas as técnicas de evolução de esquema de dados abordadas na Seção 3.5.1 (*Criação/Modificação*, *Suplementação* e *Versionamento*) podem perfeitamente conviverem em um mesmo sistema. Isso se deve ao fato delas possuírem aspectos de aplicabilidade complementares. Essa característica é de grande serventia para os DBA, pois aumenta a utilidade e combinação das técnicas para resolver diversas situações de evolução presentes em seu dia-a-dia de trabalho.

Já as estratégias de propagação abordadas na Seção 3.5.2 (*Copying*, *Updating*, *Screening*, *Versioning* e *Materializing*) apresentam algum tipo de deficiência e comprometimento da velocidade de execução das operações de manipulação das instâncias. Por este motivo, não é recomendado que um SGBD tenha como base uma única estratégia para propagar suas mudanças, sob pena de ter graves problemas de desempenho e conseqüente diminuição da qualidade de seus serviços.

## 3.6 Evolução de Esquemas em Sistemas Existentes

Evolução de esquemas tem sido uma área de pesquisa ativa por muito tempo, porém devido à sua crescente necessidade, muitos trabalhos só vieram a ser produzidos recentemente. A produção de trabalhos de pesquisa sobre o tema se intensificou depois da popularização dos bancos de dados relacionais e do surgimento dos primeiros bancos de dados orientados a objeto, onde a necessidade da evolução de esquemas recebeu uma atenção especial. Podemos encontrar na literatura muitos trabalhos com os mais diversos contextos, por exemplo: evolução de esquemas em banco de dados relacionais e orientados a objeto, evolução de esquemas XML, evolução baseada em ontologias, evolução de *software*, evolução baseada em *workflows*, gerenciamento de versões, ge-

[6] http://www.oracle.com

renciamento de modelos e mapeamento, entre outras.

O trabalho de Roddick (1992) traz uma bibliografia sobre evolução de esquemas contemplando mais de cinqüenta trabalhos sobre o assunto. Já o trabalho de Rahm e Bernstein (2006) traz uma vasta bibliografia *on-line* sobre evolução de esquemas, disponível na *Internet* através de uma ferramenta de categorização de conteúdo, chamada Caravela [7] [Aumüller & Rahm, 2006]. Essa ferramenta reúne e categoriza um grande número de publicações sobre evolução de esquemas e se encontra na sua versão 2.0. Vale ressaltar que nesta ferramenta estão disponíveis a maioria dos trabalhos contidos na bibliografia levantada por Roddick (1992) e muitos outros. Para se ter uma idéia, em outubro de 2006, mais de trezentos trabalhos sobre evolução de esquemas e áreas relacionadas estavam categorizados.

Dados os diversos contextos dos trabalhos existentes na área de evolução de esquemas, descritos anteriormente, daremos maior atenção aos relacionados a banco de dados relacionais, pois fazem parte do escopo deste trabalho. Adiante, veremos os principais trabalhos relacionados divididos em dois grupos: as iniciativas acadêmicas e comerciais. Ao final do capítulo, analisaremos criticamente cada uma delas.

### 3.6.1 Iniciativas Acadêmicas

*DB-Main* [8] *(Database Maintenance and Evolution)* [Hainaut et al., 1994] é uma ferramenta *CASE* [9] genérica, dedicada ao desenvolvimento de aplicações de banco de dados. Possui muitas facilidades para a geração de documentação e recursos para realizar rastreabilidade/propagação de mudanças. Foi projetada para ajudar os desenvolvedores de aplicações de banco de dados a realizarem tarefas como: engenharia reversa, reengenharia, migrações, integrações, manutenções e principalmente evolução de esquemas. Desenvolver aplicações de banco de dados é uma tarefa complexa, por esse motivo torna-se necessário o uso de uma série de ferramentas de apoio ao processo de desenvolvimento. Muitas dessas ferramentas são de cunho muito específico (ex. ferramenta para edição gráfica de esquemas conceitual/lógico e geração de código). A *DB-MAIN* possui várias dessas ferramentas que trabalham de forma integrada, facili-

[7] http://se-pubs.dbs.uni-leipzig.de
[8] http://www.db-main.com
[9] *Computer-Aided Software Engineering*

tando o trabalho do desenvolvedor. Uma destas ferramentas é dedicada à evolução de esquemas.

Hick e Hainaut (2002) desenvolveram um trabalho que aborda a evolução de banco de dados do ponto de vista do desenvolvedor e propõem uma estratégia geral que mostra como as mudanças de requisitos podem ser propagadas para os esquemas de dados, instâncias e programas, de forma otimizada. Para a validação dessa estratégia, foi desenvolvido um protótipo que auxilia a realização da evolução de esquemas em banco de dados relacionais. Esse protótipo foi desenvolvido como uma extensão da *DB-MAIN* para dar suporte à estratégia proposta. Ele tem a capacidade de gerar automaticamente o código fonte correspondente a alguma seqüência de transformações a serem realizadas no banco para atender às mudanças de especificação de requisitos. Além disso, também gera relatórios específicos, que são utilizados pelos desenvolvedores para a identificação dos trechos do código fonte dos programas que necessitam ser modificados como conseqüência da evolução. O protótipo foi desenvolvido utilizando a linguagem de programação imperativa *Voyager 2* [Englebert, 2004], que foi definida e implementada para uso específico da ferramenta *DB-MAIN*.

*MeDEA (Metamodel-based DB Evolution Architecture)* [Dominguez et al., 2008] é um trabalho desenvolvido pelo Departamento de Informática e Engenharia de Sistemas da Universidade de Zaragoza, na Espanha. Consiste numa arquitetura genérica de apoio à evolução de esquemas que permite manter a rastreabilidade entre os diversos artefatos envolvidos no processo de desenvolvimento de um banco de dados. Através dela é possível relacionar mudanças ocorridas nas diferentes fases do ciclo de vida de uma base de dados, desde os estágios iniciais do projeto, até a fase de implantação e manutenção. Dessa forma, a consistência entre os diferentes níveis do projeto do banco de dados (ver Seção 3.3) é devidamente tratada. Se uma mudança é feita no nível conceitual, por exemplo, ela é propagada em todos os artefatos dos outros níveis, inclusive nas instâncias de dados, para corresponderem à mudança ocorrida.

A arquitetura faz uso de um artefato estrutural que é aplicado quatro vezes para armazenar, respectivamente, o conhecimento da modelagem conceitual, o processo de tradução, o conhecimento da modelagem lógica e das instâncias de dados. É importante notar que assim como no projeto de um banco de dados, a arquitetura tem três níveis de abstração. Os esquemas dos componentes estão situados no nível de maior grau de

abstração (meta-modelo) e a base de conhecimento no de mais baixo (dados). O restante dos elementos estão situados no nível de modelo, conforme ilustrado na Figura 3.4.

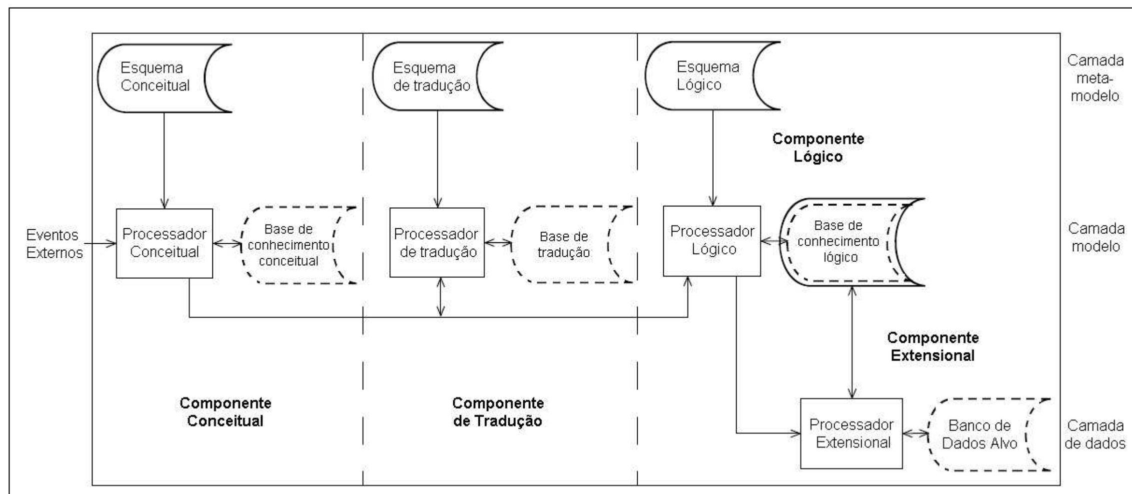



**Figura 3.4** Arquitetura para evolução de BD. Fonte: adaptado de [Dominguez et al., 2008].

O componente conceitual captura o conhecimento do mundo real (esquema conceitual), o componente lógico captura o conhecimento descrito nas estruturas de dados (esquema lógico), o componente extensional captura o conhecimento da linguagem usada na implementação e o componente de tradução grava todas as informações necessárias para que as mudanças realizadas no esquema conceitual sejam automaticamente refletidas no esquema lógico. Esse processo de tradução consiste na aplicação de regras de transformações, que não produzem somente um modelo novo, mas também um conjunto de traduções elementares, que são gravadas na base de conhecimento. O objetivo não é somente especificar como os elementos conceituais são traduzidos em elementos lógicos, mas também o processo de tradução. Dessa forma, armazena-se o caminho da tradução, tornando-se possível manter a rastreabilidade/propagação das mudanças de uma maneira mais eficiente.

Outro trabalho relacionado é a metodologia *PISE (Program Independency Schema Evolution)* [Ra, 2004], que foi concebida com o objetivo de amenizar o problema do impacto causado pelas evoluções no esquema de um banco de dados sobre aplicativos que dele dependem. O mecanismo de evolução da metodologia *PISE* é baseado em visões, ou seja, cada programa acessa o banco de dados através de uma visão de esquema diferente a ele atribuída. São previstos dois tipos de evolução de esquema: com aumento

de capacidade (ex. adição de atributo) ou sem aumento de capacidade (ex. renomeação de um atributo). É importante frisar que a metodologia *PISE* funciona supondo que todos os esquemas utilizados por usuários humanos ou aplicativos são visões derivadas de um mesmo esquema subjacente.

Existe uma grande variedade de trabalhos relacionados a Evolução de Esquemas e Propagação de Mudanças na área de Banco de Dados Orientados a Objetos que utilizam a técnica de versionamento. Como essa área e técnica não fazem parte diretamente da solução proposta nesse trabalho, mas estão relacionados a ela de forma complementar, vamos explanar sem muitos detalhes algumas iniciativas [de Matos Galante, 2003]. Isso trará um entendimento melhor sobre o que existe no contexto de evolução de esquemas e propagação de mudanças de uma maneira mais geral.

1. O *COAST (Complex Object And Schema Transformation)* [Lautemann, 1999] é um sistema que define um mecanismo para gerenciamento da evolução de esquemas em bancos de dados orientados a objetos [Lautemann, 1997]. O principal objetivo é manter as aplicações sempre em execução, garantindo dessa forma a integridade do esquema. Utiliza funções de conversão para atualizar o banco de dados em decorrência de uma modificação. A respeito das estratégias de conversão, o usuário pode optar entre a conversão imediata ou adiada;
2. O *DBEvolution* [Benatallah, 1999] é um framework implementado através de um método híbrido para gerenciamento da evolução de esquemas, combinando as técnicas de modificação e de versionamento. Seu processo de propagação de mudanças é realizado de forma imediata, logo após a modificação no esquema, podendo ser realizado por modificação, versionamento ou emulação;
3. O *Farandole 2* [Al-Jadir & Leonard, 1998] define um mecanismo baseado em versões para gerenciar a evolução de esquemas. Ele apresenta um aumento relativo de desempenho, à medida que os objetos não são copiados, nem propagados na presença de modificações de esquemas, apenas multi-objetos são definidos, evitando a necessidade de definição de regras complexas de propagação;
4. O *Sades Evolution (Semi-Autonomous Database Evolution System)* [Rashid, 2001] apresenta um estudo sobre evolução de esquemas do ponto de vista de linguagens de programação. O mecanismo de evolução está centrado nas modificações realizadas nas definições das classes, gerando diferentes visões para o esquema conceitual. Este

sistema propõe um método flexível para adaptação das instâncias durante o versionamento das classes.

### 3.6.2 Iniciativas Comerciais

A Oracle [10] vem desenvolvendo alguns recursos para tratar questões sobre reorganização de dados. Segundo Sockut e Goldberg (1979), o termo reorganização de dados vem sendo gradativamente substituído ou incorporado semanticamente por evolução de esquemas de dados, porém o trabalho *Oracle Database 10g Release 2 On-line Data Reorganization & Redefinition* [Oracle, 2005] ainda utiliza o termo reorganização de dados.

Desde a versão 9i, a Oracle vem introduzindo recursos de redefinição de dados de forma *on-line* (sem tirar os serviços do SGBD do ar, ver Seção 3.4). Isso permite aos usuários o completo acesso ao banco de dados mesmo durante a reorganização dos dados, aumentando a disponibilidade e diminuindo as paradas programadas. Dessa forma, os administradores podem realizar não somente a reorganização de dados, mas também redefinição deles de maneira *on-line*. O Oracle 10g já dá suporte a muitas características *on-line*: adição de *constraints*, *triggers* e *partitions*; criação e *rebuilding* de índices; transposição de tabelas sem a necessidade de recriar os índices novamente; adição, remoção, renomeação de atributos; mudança de tipo de dados e várias outras. Todas essas mudanças caracterizam evolução de esquemas.

Assim como a Oracle, a IBM [11] também vem desenvolvendo recursos para tratar evolução de esquemas e propagação de mudanças de forma *on-line*, com o objetivo de reduzir a necessidade de tirar os serviços do banco de dados do ar para realizar evoluções. Esses recursos começaram a ser implementados desde o SGBD DB2 8 com a evolução de esquemas *on-line* e vêm sendo estendidos no DB2 9 com a definição de banco de dados sob demanda (DDOD) [12]. Algumas operações que podem ser feitas de forma *on-line* são: adição de colunas; renomeação de tabelas, porém uma grande quantidade de restrições relativas a *triggers* e *views* devem ser levadas em conta; mudança de índices; mudança do tipo da coluna de dados; adição de partições ao final da *table*

[10] http://www.oracle.com

[11] http://www.ibm.com

[12] Database Definition on Demand

*space*, dentre outras [Coleman, 2007].

Assim como nas iniciativas acadêmicas, existem também algumas iniciativas comerciais que abordam evolução de esquemas e propagação de mudanças na área de Orientação a Objetos fazendo uso da técnica de versionamento. Todas essas iniciativas foram implementadas em SGBD comerciais: *Jasmine* [Jasmine, 2003], *POET* [Poet, 1997], *Versant* [Versant, 1997], *GemStone* [GemStone, 2003], *Itasca* [Itasca, 1995], *ObjectStore* [ObjectStore, 2003], *O2* [O2, 1999] e *ObjectivityDB* [Objectivity, 2001]. Não explanaremos as iniciativas citadas, pois elas não são foco do presente trabalho.

## 3.7 Análise Crítica Comparativa

Nesta seção analisaremos as principais características das iniciativas descritas anteriormente. Não serão levadas em conta as iniciativas da área de Banco de Dados Orientados a Objetos. Como comentado na Seção 3.2.2, mudanças de requisitos são traduzidas em mudanças de especificação que correspondem a algum nível de abstração do projeto do banco de dados (conceitual, lógico ou físico). De forma a garantir a consistência, se necessário, essas mudanças devem ser propagadas para os outros níveis de abstração do projeto. Devido à complexidade desse processo, ele deve ter suporte nas ferramentas que o apóiam. Esse é o principal requisito que as ferramentas analisadas devem atender. Para a comparação, utilizamos seis critérios importantes para um sistema que visa fornecer autonomia na tarefa de evolução de esquemas, são eles:

1. **Atua em ambientes heterogêneos e distribuídos** - Permite verificar a capacidade do sistema de atuar em diversas plataformas de SGBD simultaneamente (heterogeneidade), considerando ambientes de banco de dados distribuídos;
2. **Possui base de conhecimento** - Permite verificar a capacidade do sistema de armazenar conhecimento sobre todo o processo de evolução de esquemas realizado, com o objetivo de facilitar evoluções futuras, realizar análises estatísticas ou tomar decisões para melhoria de desempenho;
3. **Utiliza única versão do esquema** - Permite verificar se o sistema considera apenas uma única versão do esquema evoluído, descartanto o anterior (não utiliza versionamento). Este critério é mais utilizado no contexto de banco de dados relacionais;

4. **Propaga atualizações automaticamente** - Permite verificar a capacidade do sistema de propagar mudanças de esquemas nos vários níveis de abstração de um projeto de banco de dados, ou em diversas plataformas de SGBD existentes;
5. **Possui arquitetura multi-agente** - Permite verificar a capacidade do sistema em ser distribuído e de tolerar falhas; e
6. **Gera código fonte** - Permite verificar a capacidade do sistema de gerar código fonte contendo toda a sequência necessária para a evolução de um determinado esquema (*scripts*), ou código fonte de elementos relacionados ao esquema (ex. *triggers* e *packages*), já contemplando as mudanças relativas à evolução.

A ferramenta *DB-MAIN* utiliza um modelo de Entidade-Relacionamento Genérico (GER) para desenhar os esquemas do banco de dados. Este modelo nos permite representar uma grande variedade de conceitos vindos de diferentes níveis de abstração e de diferentes modelos específicos. Um esquema conceitual desenhado em *UML* [13], por exemplo, pode ser representado com os mesmos conceitos genéricos que um esquema *Oracle* no nível físico. Na *DB-MAIN* dois esquemas equivalentes podem ser mapeados por uma série de transformações reversíveis que preservam a semântica, permitindo uma transformação fiel do esquema de um nível de abstração para o outro (ex. lógico para físico). Todas essas transformações podem ser acompanhadas através de *logs* de processamento. O armazenamento e gerenciamento do histórico de transformações também é feito. Os projetistas de banco de dados podem trabalhar livremente, realizando todas as atividades padronizadas ou não padronizadas. Além de evolução de esquemas, a ferramenta pode ser usada também para engenharia reversa, engenharia avançada e outras aplicações.

O protótipo desenvolvido por Hick e Hainaut (2002) é capaz de gerar o código fonte com toda a seqüência de ações relativas a uma mudança, seja ela em qualquer nível do projeto do banco de dados. O código gerado deve ser revisado e aplicado no SGBD alvo pelo administrador de banco de dados de forma manual.

A arquitetura *MeDEA* [Dominguez et al., 2008], é um trabalho similar à extensão da *DB-MAIN* realizada por Hick e Hainaut (2002). O diferencial é que MeDEA grava explicitamente o conhecimento sobre o processo de tradução das modificações, não

[13] *Unified Modeling Language (UML)* é uma linguagem de modelagem não proprietária.

somente as transformações, de um nível do projeto para o outro. Do mesmo modo, a informação sobre os níveis conceitual, lógico e físico é armazenada para reforçar a proposta de propagação das mudanças. Os dados são armazenados em uma base de conhecimento como maneira de garantir a rastreabilidade do processo de tradução, evitando recalcular os elementos lógicos que resultam de elementos conceituais não modificados, sendo desnecessária uma nova aplicação do algoritmo de tradução a partir do zero. Isto é importante não só para evitar desperdício de recursos computacionais, mas também porque o administrador de banco de dados tem de tomar decisões apenas em relação aos elementos modificados. A implementação de MeDEA também gera código fonte com a seqüência de ações relativas a uma mudança. Ela considera apenas uma única versão do esquema a ser mantida num ambiente de banco de dados relacional (Oracle 10g utilizando a linguagem de programação *PL/SQL*).

A metodologia *PISE* tem uma grande aplicabilidade em SGBD relacionais. Apesar de sua abordagem ser aplicável a bancos de dados orientados a objeto, ela não é suficiente para atingir seus objetivos nesse contexto. Isso acontece porque ela não aborda operações de evolução específicas a bancos de dados orientados a objeto. Dada a quantidade mais restrita de operações com as quais *PISE* tem que lidar, a sua abordagem é mais simples que as demais citadas. Consideramos justamente essa simplicidade como seu principal ponto positivo. Como os programas acessam o banco de dados através de visões, é indiferente para eles se a mudança no esquema é real ou virtual. Assim sendo, podemos, pelos nossos critérios, considerar o mecanismo de evolução da metodologia como transparente. A metodologia *PISE* não gera nenhum tipo de código fonte de apoio.

Das iniciativas comerciais, as características de redefinição de dados *on-line* da Oracle e IBM não são realizadas de forma automática nem com um alto grau de rastreabilidade, tendo o DBA que dizer explicitamente a modificação a ser realizada num determinado nível para um determinado esquema. Porém essas características trazem benefícios muito importantes como aumento do desempenho do banco de dados, tempo de resposta e melhor utilização de espaço em disco.

Sintetizando, o Quadro 3.1 mostra o resultado da comparação dos trabalhos analisados, considerando os seis critérios descritos anteriormente.

**Quadro 3.1** Comparativo dos sistemas de evolução de esquemas analisados.

| Iniciativas | Atua em ambientes heterogêneos e distribuídos | Possui base de conhecimento | Utiliza única versão do esquema | Propaga atualizações automaticamente | Possui arquitetura multi-agente | Gera código |
|---|---|---|---|---|---|---|
| DB-MAIN | Não | Não | Sim | Sim | Não | Sim |
| MeDEA | Não | Sim | Sim | Sim | Não | Sim |
| PISE | Não | Não | Sim | Não | Não | Não |
| Oracle *on-line* | Não | Não | Sim | Não | Não | Não |
| IBM *on-line* | Não | Não | Sim | Não | Não | Não |

## 3.8 Conclusão

Este trabalho propõe a construção de uma solução baseada em conceitos da Computação Autônoma e Evolução de Esquemas. Com relação aos conceitos apresentados neste capítulo, a solução é voltada ao problema da evolução do esquema lógico. Ela não pretende somente aplicar modificações, mas fazer isso de forma inteligente, controlando a concorrência e dependência dos processos de atualização no nível lógico, interagindo com o SGBD, garantindo a consistência e integridade dos dados armazenados. A implementação da solução respeitará os cinco níveis de complexidade de evolução de esquemas, preservando assim a consistência do banco de dados e fará alterações de forma *on-line*. Isso porque no contexto na qual será implementada não é possível interromper serviços oferecidos pelo SGBD para que um banco seja alterado, pois se tratam de serviços essenciais. Neste caso, parar um banco para ser alterado, significa parar todos os serviços dependentes dele, o que causaria um enorme impacto e custo. Manter o banco de dados a ser alterado em operação enquanto o processo de evolução de esquemas é realizado, requer o cuidado de evitar que as aplicações executando naquele momento não sejam invalidadas, ou seja, é necessário controlar esse tipo de cenário, para ser mantida a integridade do banco e o correto funcionamento das aplicações.

Assim como as implementações das soluções *DB-MAIN* [Hick & Hainaut, 2002], *MeDEA* [Dominguez et al., 2008] e *PISE* [Ra, 2004], a implementação da solução proposta neste trabalho atua num ambiente de banco de dados relacional e considera que existe apenas um único esquema ativo a ser modificado, ou seja, não aborda multiversões. A implementação também utiliza uma base de conhecimento para gravar as transformações realizadas, em forma de eventos de atualização. O grande diferencial do atual trabalho com relação a todos os considerados neste capítulo é que ele faz a

aplicação das alterações referentes à evolução de um determinado esquema de forma automática. É capaz de propagar as atualizações de um determinado esquema onde quer que ele esteja replicado dentro do mesmo ambiente (usuários ou bancos diferentes), evitando dessa forma repetição de trabalho por parte do DBA e inconsistências nos bancos de dados e aplicações. Além disso, é capaz de gerar, também de forma automática, o código fonte de *triggers* e *procedures* de banco afetadas pelas modificações. Pode também ser implantado em ambientes distribuídos e heterogêneos (vários tipos de SGBD), pois utiliza uma arquitetura que possibilita sua distribuição, baseada em sistemas multi-agentes.

Como descrito na Seção 3.2.3, "evolução de esquemas não implica necessariamente num suporte completo para guardar o histórico de evolução por parte do sistema de banco de dados, mas somente a habilidade para alterar as definições do esquema conceitual sem a perda de informações" [Roddick, 1995]. Baseando-se nisso, a implementação da solução terá uma abordagem não temporal, ou seja, os esquemas antigos serão substituídos por novas versões, à medida que forem sendo criadas. Em particular, os dados serão atualizados de forma a se tornarem indisponíveis para o esquema anterior. Os programas aplicativos só poderão utilizar os novos dados depois de serem devidamente atualizados para condizerem com a nova situação do esquema. Para isso será utilizada a técnica de Criação/Modificação de Esquemas.

No Capítulo 4, veremos em detalhes todo o desenvolvimento da solução proposta.

# CAPÍTULO 4
# AutonomousDB

Após o estudo dos conceitos sobre Computação Autônoma e Evolução de Esquemas em banco de dados, vamos agora apresentar o processo de desenvolvimento da solução proposta neste trabalho, chamada AutonomousDB.

AutonomousDB é uma ferramenta capaz de realizar tarefas para evolução de esquemas em ambientes de bancos de dados distribuídos e heterogêneos, nos quais existem esquemas replicados. Ela evita que os DBA tenham que realizar manualmente as mesmas tarefas várias vezes em diferentes usuários de banco de dados, para garantir a consistência de todas as cópias de esquemas que sofreram mudanças. Estes usuários podem estar criados em SGBD de diferentes plataformas. Essa primeira versão do AutonomousDB não atua na evolução das instâncias de dados, mas no nível lógico. O principal objetivo desta etapa do trabalho é especificar todo o processo de desenvolvimento da solução proposta. Neste capítulo, detalharemos o AutonomousDB vendo o conceito de eventos, sua arquitetura, modelagem (casos de uso, modelo de análise, modelo de projeto e fluxo de eventos) e finalizaremos com o modelo de dados da ferramenta.

## 4.1 Eventos para o AutonomousDB

Para um melhor entendimento do AutonomousDB, é importante primeiro apresentar o conceito de evento para a ferramenta. Qualquer requisição feita por um DBA ao AutonomousDB para que realize a evolução de um determinado esquema de dados é chamada de **evento**. Existem diversos tipos de eventos que caracterizam evolução de esquemas, porém para esta primeira versão da ferramenta foram escolhidos quatro: *adição*, *deleção*, *renomeação* e *modificação* de atributos pertencentes a uma tabela do esquema a ser evoluído. A escolha desses quatro tipos de eventos, foi motivada por estudos que apontam um aumento médio no número dos atributos de 274% em relação à

versão inicial do esquema [Blaschka, 2000]. Também foi levado em conta o tempo hábil para a implementação da ferramenta. O número de eventos que podem ser fornecidos como entrada para a ferramenta é ilimitado. Os eventos também podem ser combinados para realizarem um plano de execução para uma evolução de esquemas. Por exemplo, se o DBA precisar criar dois novos atributos em uma tabela, levando em conta a ordem de execução dos eventos, ele pode fornecer essa ordem para o AutonomousDB. Os eventos que forem criados primeiro, serão executados primeiro. Apesar da ferramenta ter sido projetada inicialmente para tratar os quatro tipos de eventos descritos anteriormente, ela tem flexibilidade suficiente para ser estendida e incluir novos eventos.

É importante frisar que um evento só existe para o sistema a partir do momento em que o DBA preenche os campos obrigatórios na interface gráfica com os dados relativos e o adiciona. Neste momento, o sistema vai persistir esses dados passados como entrada e criar efetivamente o evento. O AutonomousDB também fornece a funcionalidade de agendar a execução de eventos no ambiente de banco de dados, de acordo com a necessidade dos DBA.

## 4.2 Especificação da solução

Nesta seção iremos especificar em detalhes a solução proposta para este trabalho, vendo aspectos de sua arquitetura, requisitos e modelagem.

### 4.2.1 Arquitetura geral

A arquitetura do AutonomousDB (mostrada na Figura 4.1) é composta de três camadas organizadas de forma modular, que permitem o isolamento das funcionalidades da aplicação. Isso facilita sua manutenção, aumenta a flexibilidade, escalabilidade e extensibilidade. A arquitetura utiliza conceitos de orientação a objetos nas camadas *Front End* e *Back End*; e orientação a agentes na Camada de Agentes, por isso, pode ser considerada híbrida. Isso pode causar alguma confusão, pois tanto objetos como agentes são entidades computacionais que encapsulam estados e se comunicam através de mensagens, mas a diferença principal é que agentes têm autonomia (não são passivos), encapsulam também comportamento e dão suporte a organizações, diferentemente

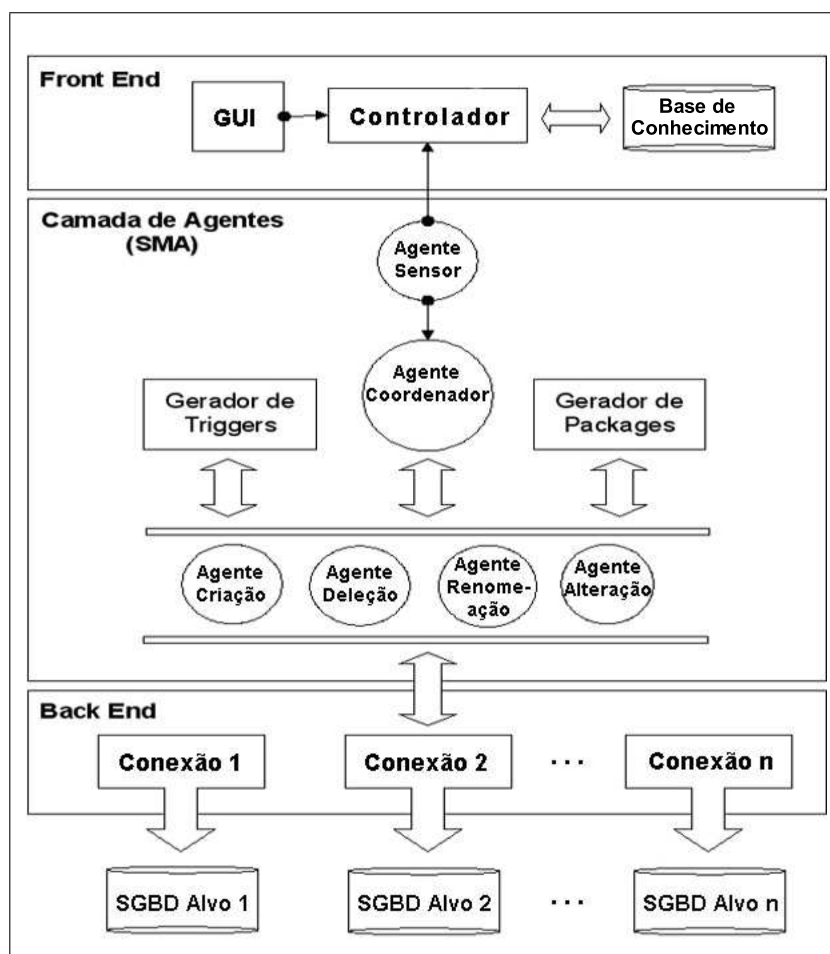



**Figura 4.1** Arquitetura do AutonomousDB.

de objetos. Adiante veremos todas as camadas em detalhes e o porquê da ferramenta utilizar uma arquitetura híbrida baseada em agentes inteligentes.

**Front End** - Esta camada é responsável pelo tratamento e armazenamento de todos os eventos fornecidos pelos DBA como entrada para o sistema. Ela é provida de uma Base de Conhecimento de uso exclusivo pelo sistema. É composta dos seguintes componentes: **GUI**, que serve de fronteira entre o sistema e o DBA, é o ponto de entrada dos dados para criação ou edição dos eventos requeridos, e o **Controlador**, que é responsável pela comunicação entre a Base de Conhecimento e o restante do sistema. O Controlador executa consultas *SQL* na Base de Conhecimento para atender requisições vindas de outros componentes (GUI ou Agentes), que em determinado momento, neces-

sitam consultar o banco de dados local e obter informações para realizar alguma tarefa. Como exemplo de requisições temos: inserir novos eventos na Base de Conhecimento, verificar se um evento específico está em erro ou se existe execução de eventos agendada para um determinado momento. Finalmente, esta camada é composta pela **Base de Conhecimento**, que é responsável por armazenar todos os dados relacionados aos eventos, informações relevantes do ambiente de banco de dados e dados de controle da própria aplicação que são críticos para o seu funcionamento. A Base de Conhecimento é consultada sempre que o sistema necessita de alguma informação para completar uma requisição ou tomar alguma decisão. É importante destacar que toda a comunicação do sistema com a Base de Conhecimento é feita exclusivamente através do componente Controlador. Isso aumenta a modularidade e segurança dos dados armazenados, pois concentra todo o acesso à Base de Conhecimento em um único ponto.

**Camada de Agentes** - Esta camada é responsável por gerenciar a execução das ações referentes aos eventos. Estas ações é que vão efetivar a evolução de esquemas específicos nos SGBD alvo. Segundo Sayão (2003), a utilização do paradigma de orientação a agentes é adequada para tratar problemas complexos ou de natureza intrinsecamente distribuída. Dado o contexto de atuação do AutonomousDB em ambientes de banco de dados heterogêneos e distribuídos, a utilização de agentes inteligentes torna-se bastante aconselhável. Isso fornece ao AutonomousDB a capacidade de ser distribuído em vários servidores e realizar operações em diversos SGBD heterogêneos. Essa capacidade possibilita também a execução das ações de forma eficiente, independente e com tolerância a falhas, mantendo um melhor controle sobre erros em tempo de execução. Por todos os motivos apresentados, decidimos por utilizar o paradigma de Orientação a Agentes para modelagem desta camada. A abordagem de Objetos Distribuídos também poderia ter sido utilizada sem problemas, porém o paradigma de Orientação a Agentes dá um melhor suporte à modelagem de comportamentos. Esta camada é constituída por um Sistema Multi-Agentes (SMA) composto por seis agentes: um **Agente Sensor**, que verifica a existência de novos eventos ou eventos agendados para serem executados; quatro **Agentes Atuadores** (Criação, Deleção, Renomeação, Alteração), que executam ações específicas dependentes do tipo de evento (comandos DDL) nos SGBD alvo; e um **Agente Coordenador**, que é responsável por gerenciar a divisão dos eventos de acordo com o tipo, delegando ao Agente Atuador correspondente à atividade de executar um

determinado evento. O Agente Coordenador atua como uma espécie de "roteador de eventos", gerenciando também toda a comunicação entre os agentes, o *status* dos eventos executados e o registro de *log* do sistema. Como mostrado na Figura 4.1, podemos ver que existe um agente dedicado para cada tipo de evento. Os Agentes Atuadores são modulares, ou seja, podem ser adicionados ou removidos do SMA sem causar problemas de indisponibilidade do sistema como um todo.

Todos os Agentes Atuadores são capazes de gerar automaticamente o código fonte atualizado de *triggers before insert update (BIU)* e *update log packages* para as tabelas que foram afetadas por um determinado evento, já considerando as novas mudanças. Por exemplo, se um evento de criação de atributo é executado em uma tabela, as *triggers BIU* que fazem referência a todos os campos desta tabela precisam ser atualizadas com a inclusão do novo atributo. Outro caso seria, se uma tabela faz controle de histórico de atualizações, o novo atributo criado deve ser incluído no código fonte da *procedure* que realiza essa atividade, sob pena de perder o histórico de atualizações para o novo atributo. O AutonomousDB trata esses casos. A geração dos códigos fontes atualizados das *triggers BIU* e *update log packages* é feita através dos componentes **Gerador de Triggers** e **Gerador de Packages**; respectivamente. Esses componentes recuperam os novos atributos da tabela modificada através de uma consulta ao dicionário de dados do SGBD alvo, que servem de subsídio para a geração dos novos códigos fontes correspondentes.

Por razões de segurança, o código fonte gerado das *triggers BIU* e *update log packages* não são automaticamente aplicados nos SGBD alvo, diferentemente das ações previstas nos eventos. Após gerados, os códigos fonte são submetidos a uma revisão do DBA, que pode fazer ajustes antes de aprovar sua aplicação no SGBD alvo. Isso acontece porque os geradores de *triggers* e *packages* não tratam as regras de negócio num nível de granularidade alto, podendo deixar de abordar algum caso específico tratado pelo DBA na revisão. O custo de implementar e manter geradores de *triggers* e *packages* que lidem com as regras de negócio levando em conta os mínimos detalhes é alto, principalmente porque os negócios das empresas vêm se tornando cada vez maiores e mais dinâmicos. O que os geradores do AutonomousDB fazem é seguir um padrão para gerarem os códigos fontes, que podem ser livremente alterados pelos DBA antes de serem aplicados no SGBD alvo. É importante deixar claro que os códigos fontes gera-

dos só precisam ser alterados/complementados em casos específicos, e não em todas as vezes que um evento for executado.

**Back End** - Esta camada é responsável por estabelecer as conexões com os diversos SGBD alvo. Como dito anteriormente, esses SGBD podem ser de diversas plataformas e possuírem vários usuários (*logins*) de banco de dados. A camada Back End, mais especificamente através das conexões nela existentes, permite que os Agentes Atuadores tenham acesso aos SGBD alvo e executem suas ações específicas para realizar a evolução de esquemas. A cada evento a ser executado, os Agentes Atuadores criam conexões novas com os SGBD alvo através de *login* e senha. Todos os dados necessários à criação das conexões são previamente recuperados da Base de Conhecimento via requisição ao componente Controlador. Toda execução de evento é feita através de uma conexão. Em caso de falha, a conexão retorna um código e uma mensagem de erro para o Agente Atuador responsável. Ele repassa essa informação para o Agente Coordenador, que registra o erro em um *log*. O Agente Coordenador também suspende a execução do evento que falhou, colocando-o com *status* de pendente na Base de Conhecimento, para posterior análise e possível reexecução pelo DBA.

### 4.2.2 Modelagem da aplicação

A partir de uma idéia inicial dos requisitos do sistema e sua arquitetura geral, foi utilizada a técnica de prototipagem para uma definição mais concreta dos requisitos. Esta técnica foi utilizada porque tem grande potencial para demonstrar os requisitos do sistema e reduzir a falta de compreensão quanto a eles [Ryan et al., 2001]. Isso ajuda o desenvolvedor a testar o sistema e melhorá-lo antes mesmo de estar finalizado. Construído um primeiro protótipo, foram identificados vários requisitos. Porém, oito deles foram considerados essenciais e escolhidos para compor uma primeira versão do AutonomousDB. Os oito requisitos funcionais escolhidos foram: *i) criar um novo evento, ii) executar eventos adicionados, iii) editar eventos adicionados, iv) agendar execução de eventos, v) visualizar código fonte gerado de triggers BIU e update log packages, vi) reaplicar eventos em erro, vii) visualizar log de processamento, viii) excluir eventos e agendamentos de execução adicionados.*

Pela análise dos requisitos descritos, cada um deles gerou um caso de uso distinto. A Figura 4.2 mostra o diagrama de casos de uso resultante após o processo de identificação

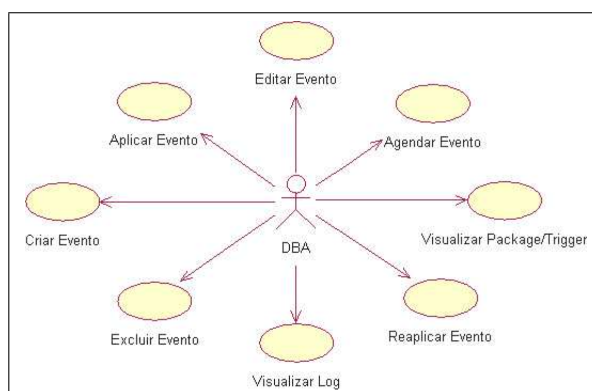



**Figura 4.2** Diagrama de Casos de Uso do AutonomousDB.

e análise inicial dos requisitos essenciais para esta primeira versão da ferramenta. Para a construção do modelo de projeto (arquitetura detalhada), que é o artefato final da especificação da solução, todos os oito casos de uso foram detalhados e realizados. Como mostrar esse processo para todos eles tornaria o trabalho longo e não agregaria tanto valor ao entendimento geral da ferramenta, foram escolhidos os dois casos de uso considerados mais importantes para serem detalhados. Os casos de uso i) e ii) são críticos para a ferramenta e o detalhamento deles ilustra perfeitamente todo o processo. O fato de detalhar apenas dois dos oito casos de uso não prejudica o entendimento da solução como um todo.

Identificados os casos de uso, o próximo passo foi especificá-los, detalhando seus fluxos de eventos principais, alternativos, pré-condições e pós-condições. Essa especificação é importante para a correta identificação das classes que irão compor o sistema, como elas estarão organizadas e como vão interagir umas com as outras.

**Especificação do caso de uso i) Criar um novo evento**

1. Objetivo: Criar um novo evento/requisição de adição, deleção, renomeação ou alteração de atributos para uma determinada tabela pertencente a um esquema específico.
2. Ator: DBA (*Administrador de banco de dados*).
3. Pré-Condições: nenhuma.
4. Fluxo Principal:
   4.1 O DBA informa todos os dados necessários para a criação do evento (tipo, tabela, banco, esquema) através da interface gráfica do sistema e confirma a ação

adicionando o evento;

4.2 O sistema cria efetivamente o "objeto evento" e o armazena numa estrutura de dados interna;

4.3 O caso de uso finaliza.

5. Fluxo Alternativo:

5.1 No item 4.1, se o DBA não informar corretamente os dados, uma mensagem de erro será exibida pedindo para que informe-os novamente.

6. Pós-Condição: O evento deve ter sido criado e armazenado corretamente numa estrutura de dados específica do sistema (*array*).

**Especificação do caso de uso ii) Executar eventos adicionados**

1. Objetivo: Executar novos eventos criados.
2. Ator: DBA (*Administrador de banco de dados*).
3. Pré-Condições: Todos os eventos devem estar corretamente criados e armazenados numa estrutura de dados específica do sistema (*array*).
4. Fluxo Principal:

   4.1 O DBA solicita a execução dos novos eventos através da interface gráfica do sistema (via acionamento de botão);

   4.2 O sistema recupera todos os eventos criados que estão armazenados na estrutura de dados interna;

   4.3 O sistema verifica para cada evento, onde o esquema especificado pelo DBA está replicado no ambiente, e grava um evento para cada replicação na Base de Conhecimento (SGBD local à parte);

   4.4 O sistema verifica a existência de novos eventos cadastrados na Base de Conhecimento e os recupera para execução;

   4.5 O sistema executa os novos eventos nos SGBD alvo e registra o *log*;

   4.6 O caso de uso finaliza.

5. Fluxo Alternativo:

   5.1 No item 4.2, se não existirem eventos novos criados e armazenados na estrutura de dados, uma mensagem de erro será exibida pelo sistema solicitando ao DBA que crie no mínimo um evento;

   5.2 No item 4.4, se não existirem eventos novos, o sistema verifica novamente a cada 5 segundos;

5.3 No item 4.5, se o evento não for executado com sucesso, o sistema registra erro no *log* e coloca o evento como pendente na Base de Conhecimento.

6. Pós-Condição: O evento deve ter sido executado com sucesso e o esquema alvo evoluído, conforme ação específica do evento.

Após a especificação dos casos de uso, o próximo passo foi realizá-los. A finalidade da realização de casos de uso é separar os interesses dos especificadores do sistema (representados pelo modelo de casos de uso e pelos requisitos do sistema) dos interesses dos projetistas (modelo de projeto). Seguindo o mesmo raciocínio anterior, vamos mostrar todo o processo de realização somente para os dois casos de uso escolhidos. Como eles representam funcionalidades críticas que trabalham de forma interligada, vamos realizá-los de forma conjunta.

**Realização dos casos de uso i) Criar um novo evento e ii) Executar eventos adicionados**

Na realização dos casos de uso primeiro foram identificadas as classes de análise, que representam um esboço das classes do sistema e são um "primeiro passo" nas principais abstrações que ele deve tratar. Junto com as classes de análise foram identificados também operações e atributos iniciais, e então, construídos os diagramas de seqüência, que mostram como as mensagens entre os objetos são trocadas no decorrer do tempo para a realização das operações. Por fim, com base nos diagramas de seqüência, foi feita a organização das classes de análise identificadas, modelando a melhor interação entre elas para atingir os objetivos das funcionalidades.

Analisando o fluxo de eventos principal nas especificações dos casos de uso i) e ii), foram identificadas as seguintes classes de análise e operações:

1. **TelaAdicionarEvento** - Representa a interface (fronteira) por onde o usuário fornece todos os dados necessários para a criação de um novo evento. Para esta classe foi identificada a operação *adicionarNovoEvento()*, que permite ao usuário efetivar a criação do evento para o sistema;
2. **ControladorEvento** - Representa a classe de controle que gerencia a inserção, recuperação e verificação de novos eventos junto a Base de Conhecimento (SGBD local). Isso é feito através das operações: *inserirNovosEventos(), retornaEventosNovos() e temRegistroNovo()*, respectivamente;

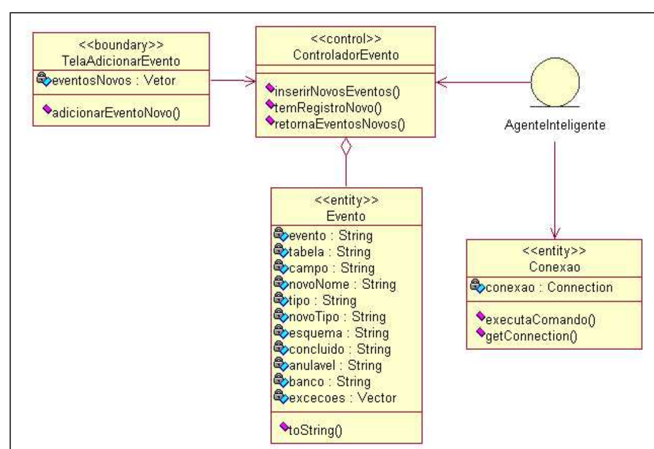



**Figura 4.3** Diagrama de Classes de análise para os Casos de uso i) e ii).

3. **AgenteInteligente** - Representa um agente inteligente atuador, responsável pela execução das ações que vão concretizar os eventos nos SGBD alvo. Essa classe tem algumas peculiaridades na sua modelagem que serão discutidas posteriormente quando apresentarmos o modelo de projeto;
4. **Evento** - Classe básica que representa um evento para o sistema; e
5. **Conexao** - Classe que representa uma conexão com um determinado SBGD alvo e executa comandos nele através da operação *executaComando()*.

A Figura 4.3 representa o diagrama de classes de análise com os detalhes das interações para os dois casos de uso. O diagrama mostra as operações e atributos identificados para cada caso. A Figura 4.4 representa o diagrama de seqüência integrado dos dois casos de uso. Ele modela a interação entre os objetos para atingir os objetivos de criar e executar um evento. É importante lembrar que todo esse processo de realização foi executado para os oito casos de uso representados na Figura 4.2, mas que por questões didáticas e de espaço, escolhemos apenas os dois principais para explanar.

Classes de análise podem ser vistas como "rascunhos" das classes de projeto. Depois de identificadas, as classes de análise são mais detalhadas e dão origem às classes de projeto, que são mais completas. Nesse processo de detalhamento, alguns objetos e interações antes identificados nas classes de análise, podem ser eliminados ou darem origem a outros nas classes de projeto. Dessa forma o diagrama de classes de análise evolui e dá origem ao modelo de projeto (arquitetura detalhada), que é mais rico em

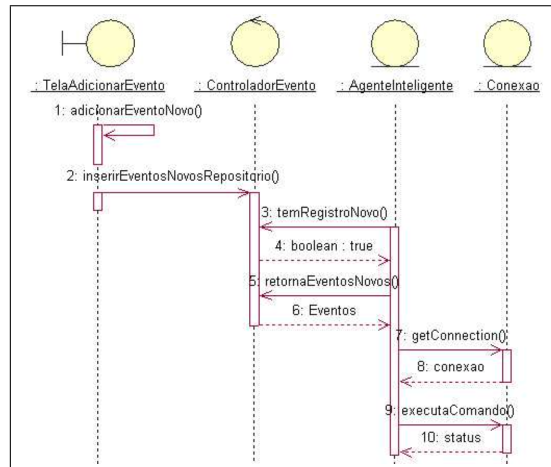



**Figura 4.4** Diagrama de seqüência para os Casos de uso i) e ii).

detalhes e por isso menos abstrato. O modelo de projeto é um artefato fundamental para a implementação de um sistema.

Com relação ao AutonomousDB, depois de construído o diagrama de classes de análise contemplando todos os casos de uso da Figura 4.2, ele foi detalhado para um nível de granularidade bem maior, por meio do qual foram identificadas novas classes, interações, operações mais específicas e atributos. A Figura 4.5 ilustra o modelo de projeto do AutonomusDB, resultado final de todo o processo de detalhamento das classes de análise. Como foi explicado na Seção 4.2.1, a arquitetura geral do AutonomousDB é híbrida, composta por objetos e agentes inteligentes. Por esse motivo, o modelo de projeto é composto por classes que quando instanciadas, representam objetos ou agentes. Adiante, vamos explicar brevemente cada uma delas e suas peculiaridades. Como objetos e agentes têm suas características próprias que interferem na modelagem, para um melhor entendimento vamos tratá-los separadamente, dividindo-os em dois grupos.

### Classes de projeto que representam objetos

1. **TelaPrincipal** - Representa a GUI do sistema, engloba operações de criação e execução de eventos, além de operações auxiliares. Foi derivada da classe de análise *TelaAdicionarEvento*, inicialmente identificada;
2. **Controlador** - Representa a classe responsável pelo acesso à Base de Conhecimento (SGBD local). Concentra operações de inserção e recuperação de dados. Por motivos

de segurança, facilidade de manutenção, possibilidade de reuso e modularidade, foi modelada como sendo o ponto único de comunicação entre o sistema e a Base de Conhecimento. Foi derivada da classe de análise *ControladorEvento*;

3. **Evento** - Classe básica que corresponde aos dados dos eventos persistidos na Base de Conhecimento. Ela permaneceu inalterada desde sua identificação no modelo de análise;
4. **Excecao** - Classe básica que corresponde aos dados relativos aos agendamentos de execução de eventos, assim como a classe *Evento* representa dados persistidos na Base de Conhecimento. Cada evento pode ter vários agendamentos de execução;
5. **GeradorTrigger** - Classe que corresponde ao gerador de código fonte para as *triggers BIU* das tabelas afetadas pela execução dos eventos. Possui uma operação para manipular os novos atributos das tabelas e gerar o novo código da *trigger* baseado neles;
6. **GeradorPackage** - Classe que corresponde ao gerador de código fonte para as *update log packages* das tabelas afetadas pela execução dos eventos. Ela possui operações para gerar as *procedures* que vão compor a *package*. Essas *procedures* são executadas para fazer o controle de histórico dos atributos atualizados, além de garantir que regras referenciais e de negócio sejam respeitadas;
7. **Conecta** - Classe que corresponde a uma conexão com um SGBD alvo. Ela possui operações que permitem o AutonomousDB executar comandos ou consultas no SGBD alvo, seja para concretizar eventos ou para recuperar dados necessários; e
8. **Util** - Esta é uma classe de apoio, criada para encapsular operações básicas e de uso geral por muitos componentes do sistema.

### Classes de projeto que representam agentes

Agentes inteligentes têm a propriedade de encapsular ativação de comportamento e dão suporte a estruturas coletivas, diferentemente de objetos que são passivos, obedientes (não têm poder de decisão). Isso modifica a forma de modelagem dos agentes em relação aos objetos. Torna-se necessário tratar a questão do comportamento, que vai abranger a forma de comunicação com outros agentes e suas ações a serem realizadas em busca de objetivos gerais ou específicos. No AutonomousDB, a forma de tratarmos o comportamento dos agentes foi através da criação da operação *setup()*. Todos eles têm

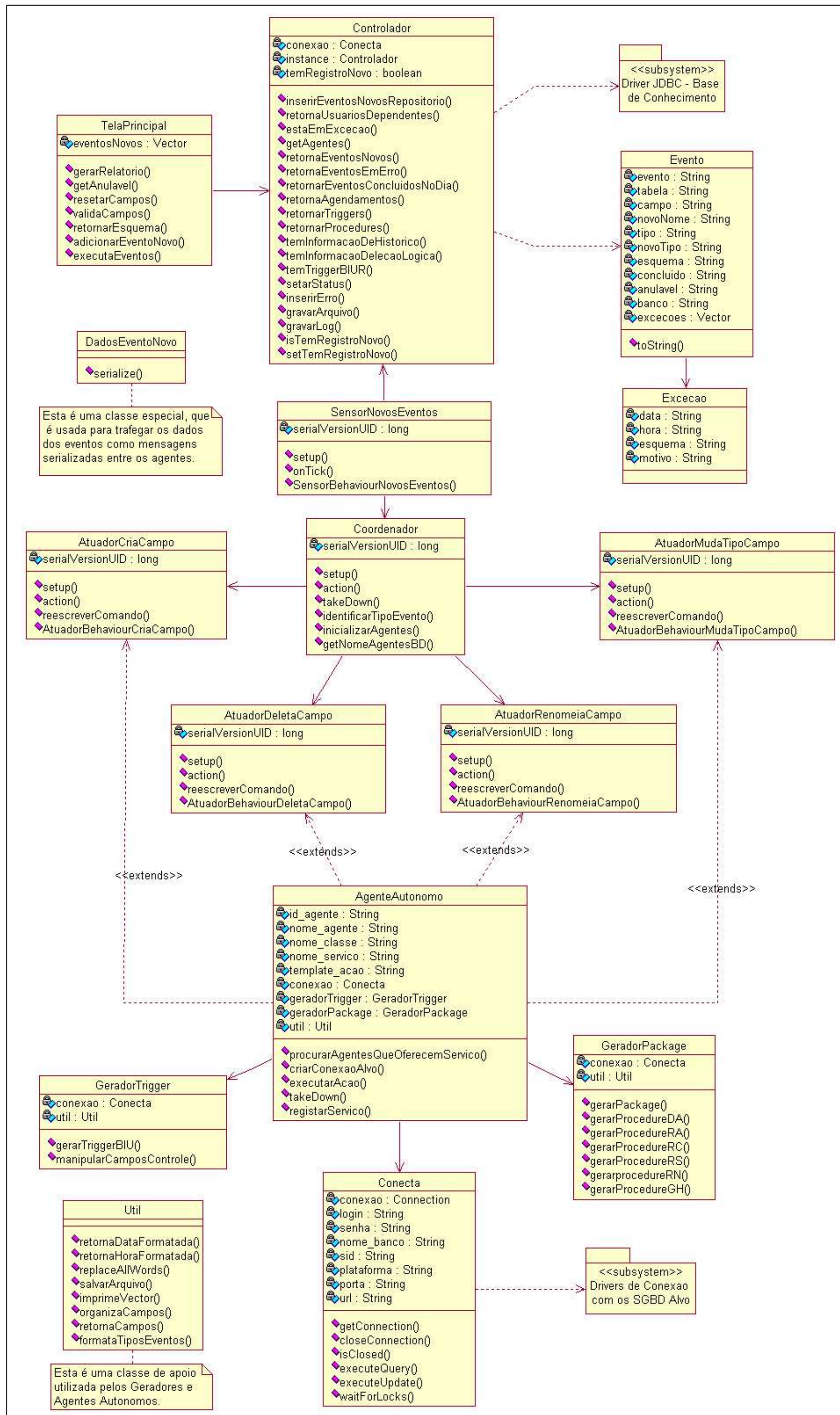



**Figura 4.5** Modelo de Projeto do AutonomousDB.

essa operação, que é responsável por associar um agente a um determinado comportamento, modelado em classes específicas. No AutonomousDB, foram modelados seis agentes que compõe o SMA da Camada de Agentes. Vejamos suas classes de projeto.

1. **SensorNovosEventos** - Responsável por verificar periodicamente junto ao componente Controlador a existência de novos eventos cadastrados na Base de Conhecimento, para serem executados imediatamente ou em outro momento (agendados). Ele faz isso através da operação *onTick()*;
2. **AtuadorCriaCampo**, **AtuadorDeletaCampo**, **AtuadorRenomeiaCampo**, **AtuadorMudaTipoCampo** - São as classes que modelam os Agentes Atuadores do sistema, responsáveis por executar ações específicas de criação, deleção, renomeação e mudança de tipo de atributos em SGBD alvos específicos. Cada Agente Atuador tem um comportamento e uma operação chamada *reescreverComando()* específicos. Ela é responsável por reescrever o *template* do comando a ser aplicado no SGBD alvo com os dados fornecidos pelo usuário e que foram previamente armazenados na Base de Conhecimento; e
3. **Coordenador** - Responsável por encaminhar os eventos para serem executados pelos agentes específicos, gerenciar a comunicação entre os Agentes Atuadores e registrar o *log* de processamento do sistema. O coordenador tem quatro tipos de comportamentos, um para gerenciar cada tipo de Agente Atuador. É através desses comportamentos que ele se comunica especificamente com um agente e toma suas decisões. O Agente Coordenador é o primeiro a ser inicializado no sistema, que através de sua operação *inicializarAgentes()* inicializa e começa a gerenciar todos os Agentes Atuadores.

No processo de criação das classes de projeto, foi observado que todos os atributos e algumas operações eram comuns a todos os Agentes Atuadores. Dessa forma, uma decisão de projeto foi criar a super classe *AgenteAutonomo*, que encapsula todos esses atributos e operações em comum. Então, cada Agente Atuador é uma especialização de *AgenteAutonomo*. Dessa forma, todas as classes de projeto que modelam os Agentes Atuadores herdam da classe *AgenteAutonomo* e implementam suas operações próprias. É importante notar que o Agente Coordenador e o SensorNovosEventos, apesar de serem também agentes inteligentes, não têm atributos e operações em comum com os Agentes Atuadores, por isso suas classes não são herdadas de *AgenteAutonomo*.

Como os agentes inteligentes se comunicam através de mensagens serializadas, foi criada uma classe de projeto chamada *DadosEventoNovo*, uma classe especial que, quando instanciada, dá origem a um objeto capaz de trafegar dados de forma serializada entre os agentes. Essa classe reúne todos os atributos das classes básicas *Evento* e *Excecao*, que por não serem serializáveis, não podem trafegar como mensagens entre os agentes.

Segundo Weiss (1999), coordenação é a habilidade de um sistema de agentes de executar alguma atividade em um ambiente compartilhado. O AutonomousDB trabalha em ambientes de banco de dados heterogêneos e distribuídos, altamente compartilhados. Em um ambiente como esse, a habilidade de coordenação para um SMA é um fator fundamental para a execução de um trabalho em conjunto. Nenhum agente inteligente de forma individual tem recursos, informação ou capacidade suficiente para resolver o problema completo. A divisão de tarefas e a resolução de conflitos deve ser realizada para prevenir o caos entre os agentes. Dessa forma, foi decidido implementar a habilidade de coordenação para o AutonomousDB através do Agente Coordenador, que tem uma visão mais ampla dos objetivos, divide as tarefas entre os Agentes Atuadores, resolve conflitos de execução simultânea de eventos em uma mesma tabela, gerencia toda a interação entre eles e faz com que o SMA respeite restrições globais.

Para finalizar o processo de modelagem do AutonomousDB, vamos apresentar o fluxo de informações que acontece entre os componentes do sistema, desde a criação de um evento até a sua efetiva execução no SGBD alvo.

**Fluxo de eventos do AutonomousDB**

1. Através da GUI do AutonomousDB, o DBA escolhe o tipo de evento que deseja criar e informa os dados pertinentes (nome da tabela, nome do atributo, banco e esquema). Informados os dados, o DBA adiciona o evento ao sistema (*adicionarEventoNovo()*). Antes de adicionar o evento, o DBA pode agendar uma execução relativa a um determinado usuário para ser executada em outro momento que não seja imediato. O DBA pode adicionar quantos eventos ou agendamentos quiser. Todos os eventos adicionados são armazenados numa estrutura de dados da própria GUI (*eventosNovos*);
2. Adicionado o evento, o DBA confirma sua execução. Com isso, o evento é enviado para o *Controlador*, que verifica em quais usuários do banco de dados o esquema

alvo está replicado dentro do ambiente (*retornaUsuariosDependentes()*). Feito isso, o *Controlador* insere um evento para cada usuário que tem o esquema alvo replicado e seus respectivos agendamentos, se existirem, na Base de Conhecimento;

3. Paralelamente aos fluxos 1 e 2, o *Agente SensorNovosEventos* verifica periodicamente (a cada 5 segundos) a existência de novos eventos ou agendamentos (*temRegistroNovo()*) cadastrados para serem executados naquele momento. Se existirem, ele recupera esses eventos (*retornaEventosNovos()*) ou agendamentos (*retornaAgendamentos*()), armazena os dados dentro de um objeto do tipo *DadosEventoNovo* e o envia (*send()*) para o *Agente Coordenador*;
4. O *Agente Coordenador* recebe a requisição e identifica de que tipo é aquele evento recebido (*identificarTipoEvento()*). Identificado o tipo do evento, o *Coordenador* encaminha-o para o Agente Atuador responsável (*send()*) por executar a ação específica no SGBD alvo e fica aguardando confirmação. O *Agente Coordenador* é capaz de tratar vários eventos de forma concorrente;
5. O Agente Atuador correspondente ao tipo de evento, quando recebe uma requisição, cria uma conexão (*getConnection()*) com o SGBD alvo para um usuário específico (os dados para o *login* são recuperados da mensagem enviada pelo *Coordenador*). Criada a conexão, o Agente Atuador monta o comando DDL a ser executado no SGBD alvo para efetivar a evolução de um esquema específico (*reescreverComando()*) e o executa (*executar()*);
6. Após a execução do comando, o Agente Atuador verifica junto ao *Controlador*, a existência de *trigger BIU* para a tabela afetada pela evolução (*temTriggerBIU()*). Se existir, ele recupera os dados referentes à *trigger* (*retornaTriggers()*) e solicita ao *GeradorTrigger* que gere o novo código fonte da *trigger* (*gerarTriggerBIU()*), baseado nas informações repassadas. O *GeradorTrigger* faz uma consulta ao dicionário de dados do SGBD alvo para recuperar os novos atributos referentes à tabela afetada (*retornaCampos()*) e junto com as informações repassadas pelo Agente Atuador, gera o novo código fonte da *trigger BIU*;
7. Gerada a *trigger BIU*, o Agente Atuador verifica junto ao *Controlador* se a tabela afetada pela evolução tem *update log package* (*temInformacaoDeHistorico()*). Em caso positivo, o Agente Atuador monta o comando que vai efetivar o evento também para a tabela de histórico (*reescreverComando()*) e o executa no SGBD alvo através

da conexão criada anteriormente;

8. Executado o comando para efetivar o evento para a tabela de histórico, o Agente Atuador solicita ao *GeradorPackage* para gerar o novo código fonte da *update log package* (*gerarPackage()*). O *GeradorPackage* solicita ao *Controlador* o nome de todas as *procedures* que compõem a *package* em questão (*retornarProcedures()*) e recupera os novos atributos (*retornaCampos()*) da tabela de histórico. Baseado nisso, é gerado o novo código fonte para a *update log package*; e
9. Gerada a *package*, o Agente Atuador fecha a conexão com o SGBD alvo (*closeConnection()*) e retorna o status para o *Agente Coordenador*, que registra o *log* do sistema (*gravarLog()*). Se em qualquer ponto do fluxo acontecer um erro, imediatamente o Agente Atuador aborta a execução para o evento que deu erro, captura os detalhes do erro e os envia para o *Coordenador*, que coloca o evento como pendente na Base de Conhecimento e armazena os detalhes do erro.

A Figura 4.6 ilustra todo o fluxo descrito anteriormente.

### 4.2.3 Modelo de persistência

Como discutido na Seção 4.2.1, o AutonomousDB possui uma Base de Conhecimento de dados para armazenar as informações relevantes aos DBA e ao próprio sistema. Após a construção do modelo de projeto (arquitetura detalhada/diagrama de classes), foi iniciado o projeto do banco de dados do AutonomousDB, que deu origem ao modelo de dados da aplicação e aos *scripts* de criação do esquema lógico. A decisão de construir o modelo de dados somente após a conclusão do modelo de projeto é justificada pelo fato de que alguns tipos de entidades só surgem nesse momento da modelagem, além do projetista do banco de dados ter adquirido um maior conhecimento da aplicação e de seu domínio. O ideal é que o modelo de dados represente e ofereça tudo que for necessário para o sistema. Com o modelo de projeto pronto, isso se torna mais factível, pois o projetista pode verificar se o modelo de dados está atendendo ou não a sua necessidade. Detalhes como prioridade no desempenho podem afetar o modelo de dados no nível lógico.

O modelo de dados concebido armazena todas as informações relacionadas aos cadastros de eventos e agendamentos, dados sobre as tabelas, *triggers* e *procedures* presentes nos SGBD alvo e dados dos próprios SGBD alvo (bancos de dados existentes,

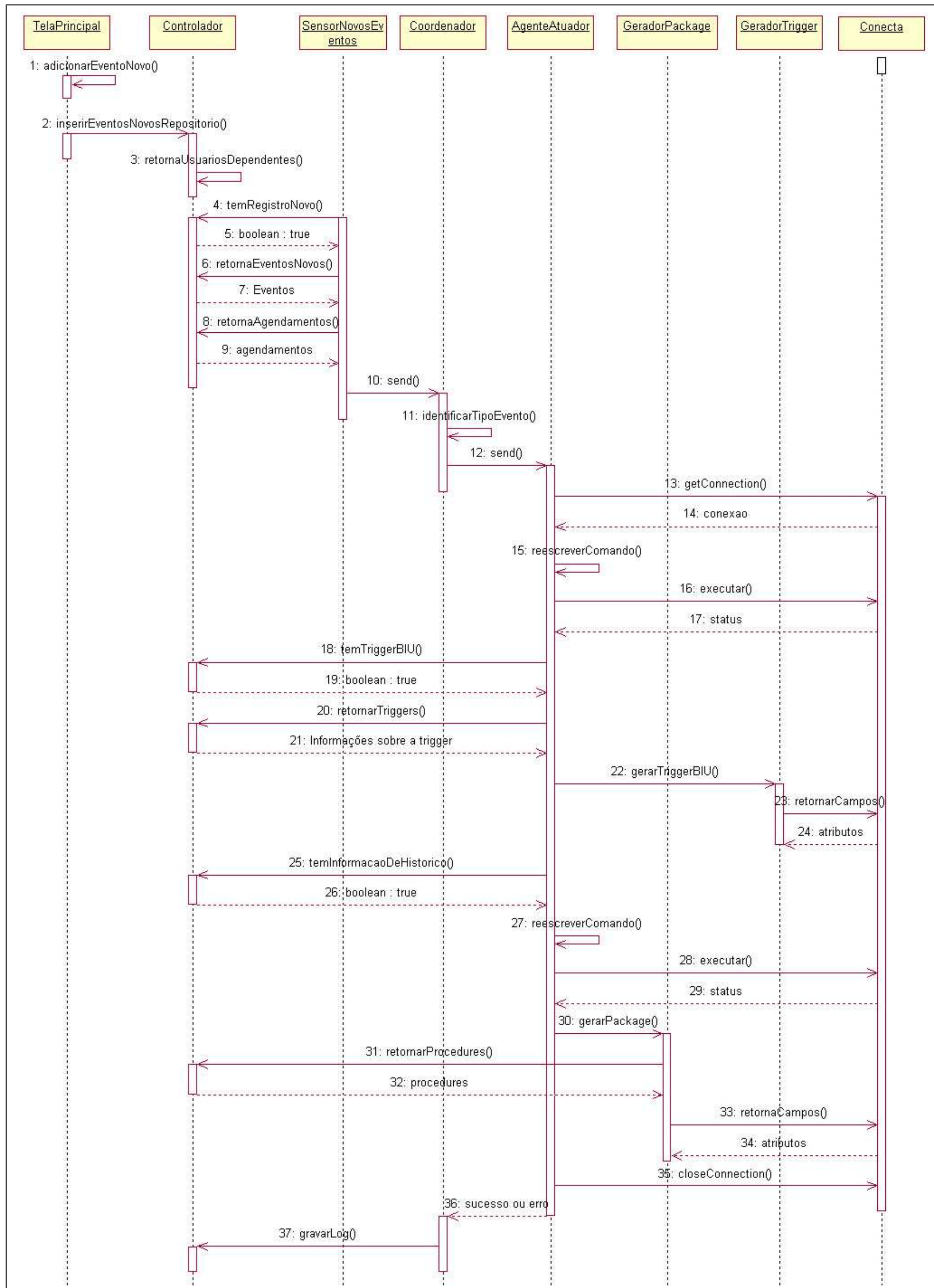



**Figura 4.6** Fluxo de eventos do AutonomousDB.

usuários, *logins* e senhas). Adicionalmente, há estruturas de armazenamento de informações específicas sobre erros de processamento que podem ocorrer e dos agentes inteligentes que compõem o sistema. A Figura 4.7 representa o modelo de dados do AutonomousDB, artefato originado após a conclusão do projeto do banco de dados, e que foi implementado na Base de Conhecimento. Adiante detalharemos as tabelas do modelo.

**Componentes do modelo de dados do AutonomousDB**

1. **Caracterização de agentes** - Cada agente inteligente deve ter seus dados básicos tais como: o serviço que oferece e a ação que executa (comando DDL), persistidos em banco. Isso facilita a adição de um novo agente ao SMA ou a modificação do serviço de agentes já existentes. A tabela *AGENTES* descreve os agentes inteligentes modelados e é utilizada pelo agente *Coordenador* para inicializar a execução do SMA. Uma vez instanciados, cada agente deve registrar seu serviço em um componente de páginas-amarelas utilizado pelo SMA. Dependendo da implementação, esse componente pode ser um agente ou uma tabela. No caso do AutonomousDB, cada agente só trata um tipo específico de evento, por esse motivo cada agente tem 1 (um) serviço;
2. **Caracterização de tabelas** - Para executar corretamente os eventos em tabelas de histórico (seguindo padrão de nomes), gerar código fonte de *triggers BIU* e *update log packages*, o AutonomousDB precisa saber quais tabelas criadas nos SGBD alvo possuem controle de histórico, quais *triggers BIU* estão relacionadas a cada tabela e quais as *procedures* relacionadas a cada *trigger*. Todas essas informações são armazenadas nas tabelas *TABELAS*, *TRIGGERS* e *PROCEDURES*, respectivamente. Se um determinado ambiente de banco de dados utiliza um padrão de nomes para os atributos de suas tabelas, *triggers* ou *procedures*, o AutonomousDB é flexível para oferecer suporte através de informações persistidas nas tabelas citadas;
3. **Caracterização de SGBD alvo** - Através das informações armazenadas na tabela *BANCOS*, o AutonomousDB consegue identificar quais usuários (e seus respectivos *login* e senha) possuem o esquema alvo de evolução replicado. Dessa forma é possível, de forma autônoma, criar um evento para cada esquema replicado no momento do cadastro. Esses eventos são cadastrados e executados, evitando que o DBA faça tudo isso manualmente;
4. **Caracterização de eventos** - Todas as informações sobre os eventos cadastrados são persistidas na tabela *EVENTOS*. Se um evento estiver relacionado a um agendamento de execução, a informação do agendamento será armazenada na tabela *AGENDAMENTOS*. Uma vez verificado um agendamento para uma determinada hora, ele será executado nessa hora. Na execução de um evento podem acontecer erros. Qualquer ocorrência de erro é registrado na tabela *ERROS*, que vincula um erro (código e mensagem) a um determinado evento e não dá sua execução como concluída.

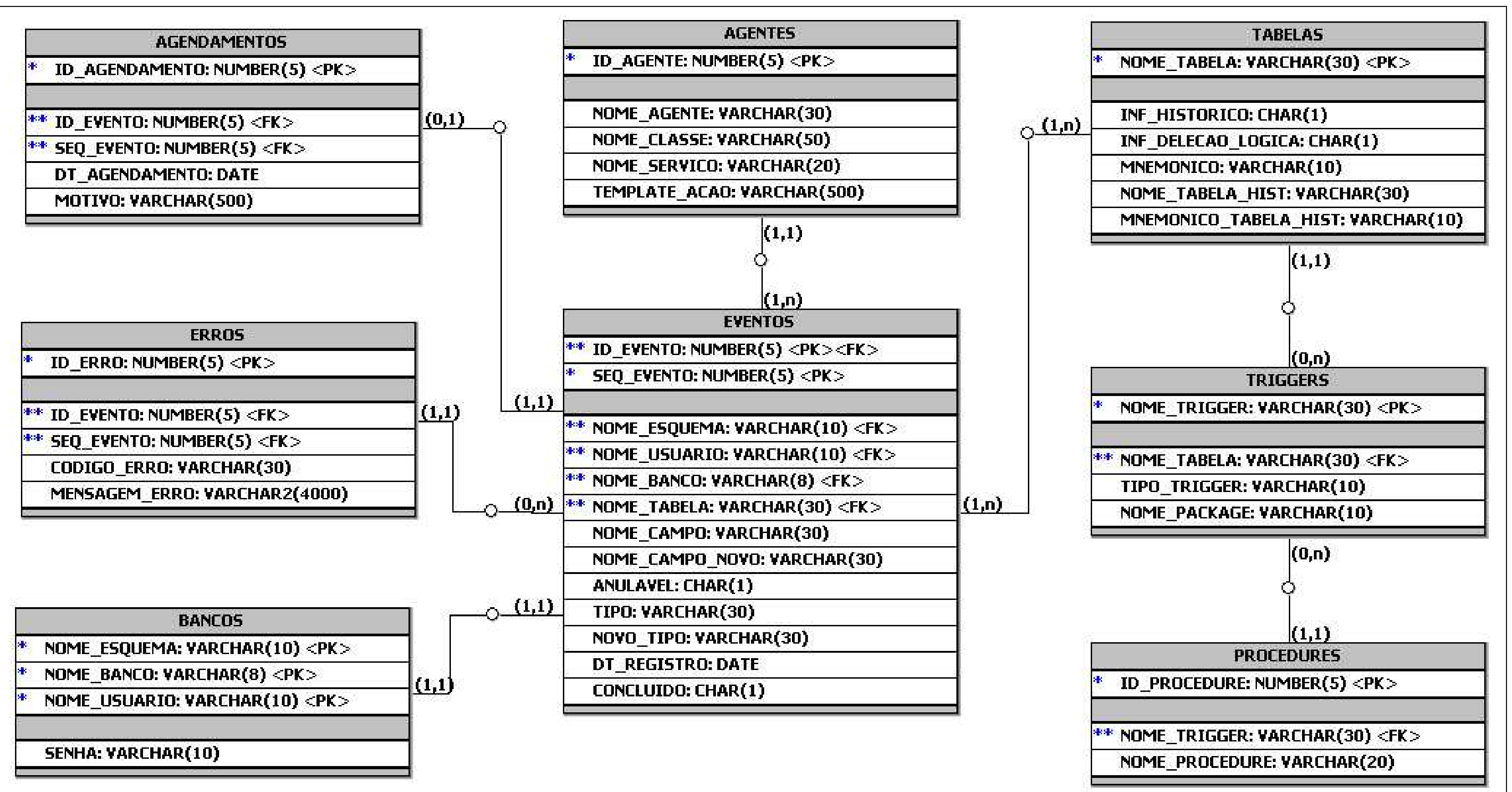



**Figura 4.7** Modelo de Dados do AutonomousDB.

## 4.3 Conclusão

Neste capítulo, foi apresentada a especificação completa da ferramenta AutonomousDB. Foram escolhidos os oito principais requisitos para comporem essa primeira versão da ferramenta. O diagrama de casos de uso foi apresentado e todo o processo de realização dos casos de uso foi explicado (especificação dos casos de uso, identificação das classes de análise, modelo de análise, diagrama de sequência, identificação das classes de projeto e modelo de projeto). Por questões didáticas e de espaço, esse processo foi mostrado na íntegra somente para os dois casos de uso mais críticos: *i) Criar um novo evento* e *ii) Executar eventos adicionados*. O modelo de projeto foi apresentado contemplando todos os casos de uso. Para um melhor entendimento do que acontece desde a criação de um evento até sua execução nos SGBD alvo, mostramos e explicamos um fluxo de eventos completo, com todas as operações executadas. Por fim, apresentamos o modelo de persistência do AutonomousDB, que está implementado na Base de Conhecimento (SGBD local).

No Capítulo 5, veremos a implementação e a aplicação do AutonomousDB em um caso real, posteriormente, mostrando seus resultados.

CAPÍTULO 5

# Uma aplicação real do AutonomousDB: o caso do SERPRO

Neste capítulo, validaremos o AutonomousDB por intermédio de sua implementação, mostrando uma aplicação real da solução especificada. Primeiramente, apresentaremos o problema escolhido e o ambiente de banco de dados no qual ele se encontra, depois explicaremos todos os detalhes da implementação para solucioná-lo. É importante frisar que toda a implementação do AutonomousDB para o caso escolhido foi guiada por sua especificação, apresentada no capítulo anterior. Por fim, faremos um experimento, apresentando a análise e discussão dos principais resultados obtidos.

## 5.1 O caso do SERPRO

Para validar o AutonomousDB, um problema que ocorre com muita freqüência no SERPRO [1] foi escolhido: o trabalho repetitivo da equipe de DBA em realizar a evolução de esquemas em ambientes de bancos de dados distribuídos e heterogêneos, nos quais existem esquemas replicados. Para melhor entender esse problema, é importante definir primeiro como estão estruturados os ambientes de banco de dados do SERPRO.

O SERPRO possui em sua infra-estrutura, 6 (seis) ambientes de banco de dados [2]: *i) produção*, que é utilizado pelas aplicações já implantadas; *ii) homologação*, que é utilizado pelos clientes para validarem os sistemas desenvolvidos ou aprovarem manutenções (corretivas, evolutivas ou adaptativas) efetivadas em sistemas já existentes, antes que sejam implantados; *iii) desenvolvimento*, que é utilizado pelos desenvolve-

[1] O Serviço Federal de Processamento de Dados (SERPRO), é uma empresa pública de desenvolvimento de soluções em informática para o governo federal.

[2] Entendemos ambiente como um conjunto de banco de dados que podem estar implementados em SGBD distintos.

dores para construir / evoluir as aplicações; *iv) testes*, utilizado pelos desenvolvedores para realizarem testes nas aplicações em desenvolvimento antes de serem homologadas; *v) treinamento*, utilizado para cursos, apresentações e demonstrações; e por fim, *vi) grande porte* que trabalha integrado com todos os outros ambientes e é usado para o armazenamento final dos dados após processamentos feitos pelas aplicações.

O ambiente de grande porte tem uma peculiaridade: ele utiliza um SGBD hierárquico (ADABAS), diferentemente dos outros ambientes, que utilizam SGBD relacionais (Oracle 10g). Pelo fato de ser hierárquico, o ADABAS não representa seus dados como tabelas relacionadas e sim como *files* (arquivos). Cada ambiente citado tem seu próprio *file* no ambiente de grande porte, possuindo identificadores próprios, e se comunica com ele por meio de uma ferramenta chamada *SQL Ada*. Essa ferramenta atua como um *middleware* que permite outras plataformas, no nosso caso Oracle 10g, acessarem o ADABAS para realizarem consultas ou executarem programas. Como o AutonomousDB foi projetado para operar com SGBD relacionais, vamos tratar apenas a plataforma Oracle 10g.

Nos ambientes descritos, as instâncias do Oracle 10g executam em servidores diferentes (máquinas distintas). Cada uma dessas instâncias pode ter definido um ou mais banco de dados e cada banco de dados pode ter vários usuários criados. Além disso, um esquema de dados de uma determinada aplicação, pode estar replicado em vários usuários. Esses esquemas são chamados de *esquemas-chave*, e são necessários para que outras aplicações possam funcionar corretamente em um usuário específico.

A Figura 5.1 ilustra um exemplo geral de como estão estruturados os ambientes de banco de dados do SERPRO e como o AutonomousDB atua. Ela mostra 3 (três) instâncias do SGBD Oracle 10g executando em 3 (três) servidores diferentes: Servidor 1; Servidor 2 e Servidor 3. Note que as instâncias do Oracle 10g que executam nos Servidores 1 e 2 têm um único banco de dados disponível cada uma (Banco 1 e Banco 2), com dois (Usuário 1 e 2) e um (Usuário 3) usuário; respectivamente. Já a instância do Servidor 3 tem dois bancos de dados disponíveis (Banco 3 e 4) com um único usuário cada (Usuários 4 e 5). O esquema-chave de uma aplicação crítica (Esquema B destacado na Figura 5.1) está replicado em todos os usuários (Usuários 1, 2, 3, 4 e 5). Se o *Esquema B* precisar ser modificado, todas as suas cópias também devem ser modificadas no ambiente, independente de onde estejam replicadas (banco e usuário). Essa

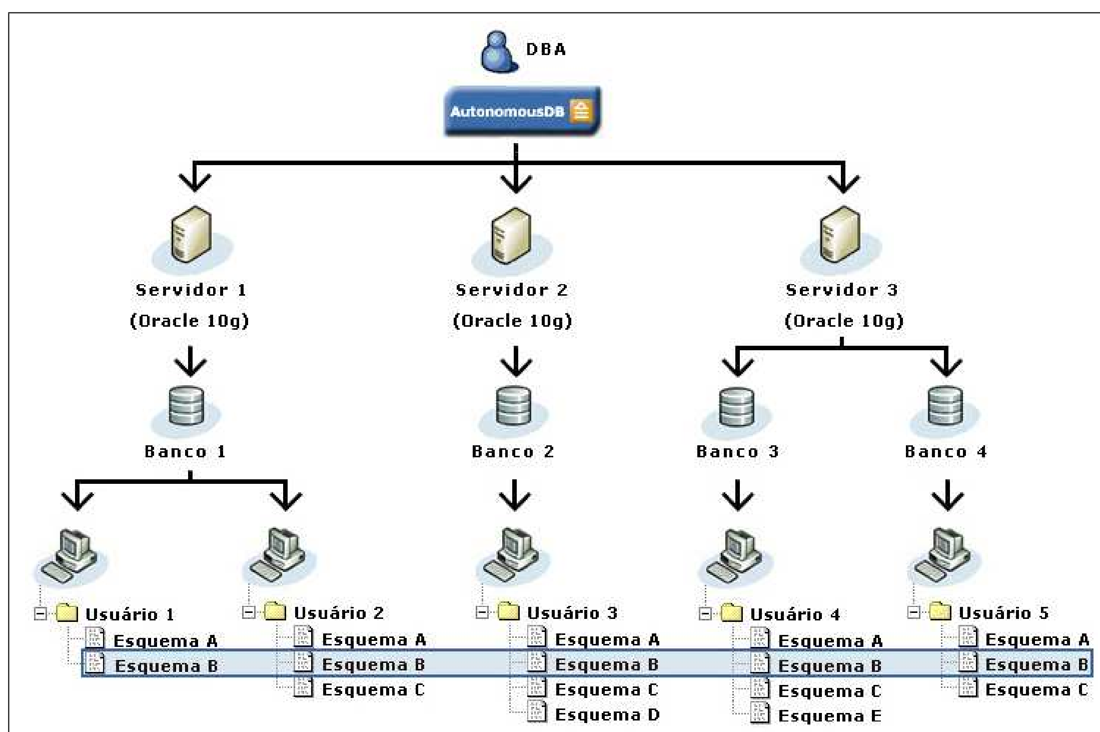



**Figura 5.1** Exemplo de como estão estruturados os ambientes de banco de dados do SERPRO.

ação é necessária para garantir a consistência do esquema que sofreu a evolução. Além disso, todas as *triggers BIU* e *update log packages* que fazem referência às tabelas do esquema modificado também devem ser atualizadas. No SERPRO, todo esse processo de atualização é feito manualmente pela equipe de DBA. Muitas vezes um esquema de um determinado usuário não pode ser atualizado de imediato, na maioria das vezes porque está sendo utilizado, o que leva os DBA a postergarem a atualização para esse caso. Como não existe uma ferramenta específica adotada, que automatize o controle das atualizações postergadas, os DBA geralmente esquecem de efetuar as mudanças no momento correto, deixando esquemas inconsistentes. Vale ressaltar que o AutonomousDB é um módulo externo ao SGBD alvo e trabalha como um "DBA virtual", realizando tarefas que antes eram feitas por um DBA humano.

Explicada a estrutura dos ambientes de banco de dados do SERPRO e a problemática da propagação de atualizações de uma forma geral, vamos agora definir o escopo do problema que será de fato alvo do experimento. As operações realizadas nos ambientes de banco de dados do SERPRO são regidas por políticas internas de segurança, que

ditam o que pode ou não ser feito. Por esse motivo, escolhemos o banco de dados *D0815* para a validação do AutonomousDB. O *D0815* é o principal banco de dados do ambiente de desenvolvimento. Nele é permitido realizar todas as operações necessárias para validar a ferramenta. Um outro fator que nos levou a escolher esse banco é o fato dele ser considerado o mais dinâmico, pois como é um banco de desenvolvimento, sofre constantes evoluções em seus esquemas devido às mudanças e complementações de requisitos das aplicações. Por esse motivo, também é o principal causador de trabalhos repetitivos e falhas humanas por parte dos DBA.

O *D0815* é o banco que dá suporte ao desenvolvimento do Sistema de Controle de Créditos (SCC), que é utilizado pela Receita Federal do Brasil (RFB), principal cliente do SERPRO. O SCC é responsável por controlar a arrecadação fiscal brasileira e trata 5 (cinco) tipos de créditos diferentes: Saldos Negativos (SNEG), Pagamento Indevido ou a Maior (PGIM), Imposto por Produto Industrializado (DIPI), Créditos Oriundos de Ação Judicial (JUDI) e PIS/PASEP Cofins (PPCF). O SCC é composto por 8 (oito) módulos (aplicações) que funcionam de forma integrada. Existe 1 (um) módulo específico para tratar cada tipo de crédito e mais 3 (três) módulos críticos: Carga PER/DCOMP (PRRC), Núcleo (CDCR) e Comunica (CCON). O Carga PER/DCOMP é responsável por carregar os dados do ambiente de grande porte (ADABAS) para o ambiente Oracle 10g. O Núcleo é responsável por fazer verificações preliminares nos dados, enquanto que o Comunica é responsável por enviar comunicações para o contribuinte. Como esses módulos trabalham de forma integrada, existem algumas dependências de funcionamento entre eles. Como estamos tratando de evolução de esquemas, vamos nos ater às dependências estruturais dos esquemas e não às dos códigos fontes das aplicações. Cada um dos módulos citados tem 6 (seis) usuários de banco de dados específicos criados no *D0815*, que são utilizados pelas respectivas equipes de desenvolvimento, totalizando 48 usuários (*logins*). O Quadro 5.1 mostra os módulos do SCC, seus respectivos esquemas, os usuários (*logins*) disponíveis e suas dependências com esquemas de outros módulos.

Pelo Quadro 5.1, vemos que para os módulos críticos do SCC, o módulo Carga PER/DCOMP precisa apenas do seu esquema (PRRC) para funcionar, o Núcleo além do seu próprio esquema (CDCR) precisa também do esquema do Carga PER/DCOMP (PRRC) replicado em seus usuários e o Comunica precisa além do seu esquema (CCON) os dos módulos Carga PER/DCOMP (PRRC) e Núcleo (CDCR). Para o restante dos mó-

Quadro 5.1 Banco D0815 do SERPRO com seus módulos, esquemas, usuários e dependências.

| Banco | Módulo do SCC | Esquema | Usuários | Dependências |
|---|---|---|---|---|
| D0815 | Carga PER/DCOMP | PRRC | prrc_ol, prrc_dl, prrc_rg, prrc_pb, prrc_bc,prrc_cg | _ |
| | Núcleo | CDCR | cdcr_ol, cdcr_dl, cdcr_rg, cdcr_pb, cdcr_bc, cdcr_cg | PRRC |
| | Comunica | CCON | ccon_ol, ccon_dl, ccon_rg, ccon_pb, ccon_bc, ccon_cg | PRRC, CDCR |
| | Saldos Negativos | SNEG | sneg_ol, sneg_dl, sneg_rg, sneg_pb, sneg_bc, sneg_cg | PRRC, CDCR, CCON |
| | Pagamento Indevido | PGIM | pgim_ol, pgim_dl, pgim_rg, pgim_pb, pgim_bc, pgim_cg | PRRC, CDCR, CCON |
| | IPI | DIPI | dipi_ol, dipi_dl, dipi_rg, dipi_pb, dipi_bc, dipi_cg | PRRC, CDCR, CCON |
| | Ação Judicial | JUDI | judi_ol, judi_dl, judi_rg, judi_pb, judi_bc, judi_cg | PRRC, CDCR, CCON |
| | PIS/PASEP Cofins | PPCF | ppcf_ol, ppcf_dl, ppcf_rg, ppcf_pb, ppcf_bc, ppcf_cg | PRRC, CDCR, CCON |

dulos, todos precisam dos esquemas do Carga PER/DCOMP (PRRC), Núcleo (CDCR) e Comunica (CCON) replicados em todos os seus usuários, além dos seus próprios esquemas para funcionarem de forma correta. Para um melhor entendimento do problema, vamos apresentar um exemplo real do que acontece no *D0815* quando é necessário evoluir um esquema que possui muitas dependências.

Imagine que uma modificação de requisitos para o módulo Carga PER/DCOMP cause a necessidade de evolução de seu esquema pela criação de um novo atributo em uma de suas tabelas. Como todos os módulos para funcionarem necessitam do esquema da Carga PER/DCOMP replicado em seus usuários (ver Quadro 5.1), uma mudança nesse esquema causa uma grande quantidade de trabalho para a equipe de DBA. Isso

se deve ao fato de ser necessário propagar a mudança em todas as cópias do esquema PRRC, atualizando tabelas de negócio, tabelas de histórico, *triggers* e *update log packages* com o novo atributo criado para garantir a consistência dos esquemas. No nosso exemplo, essas tarefas teriam de ser feitas 48 vezes (todos os usuários têm o esquema PRRC replicado). Com essa carga de trabalho repetitivo, o aumento da probabilidade de erro humano é um fato, além de consumir muito tempo do DBA, que poderia estar cuidando de tarefas mais importantes. A implementação do AutonomousDB, que apresentaremos na Seção 5.2, propõe solucionar o problema descrito, propagando as modificações de forma autônoma. Isso livra os DBA de trabalhos operacionais repetitivos, além de controlar as mudanças registrando-as, evitando falhas humanas e garantindo a consistência dos esquemas modificados que se encontram replicados no ambiente.

## 5.2 Implementação

Para a implementação do AutonomousDB, foi utilizado como referência seu modelo de projeto construído no Capítulo 4. Todas as classes de projeto e respectivas operações identificadas foram implementadas. Conseqüentemente, todos os requisitos levantados para esta primeira versão do protótipo foram contemplados. No decorrer da implementação, procurou-se seguir o modelo de projeto, o que resultou em um protótipo com um alto grau de aderência em relação à sua especificação.

Desde sua fase de levantamento de requisitos, sempre existiu a preocupação em desenvolver o AutonomousDB como um sistema multi-plataforma e escalável. Escalável principalmente no sentido de ser possível retirar ou adicionar novos SGBD alvo, sem interferir no desempenho do sistema. Por esses motivos e outros, a linguagem de programação escolhida para a implementação foi Java [Java, 2008]. Ela oferece recursos de orientação a objetos que são importantes para fornecer modularidade, escalabilidade e extensibilidade ao sistema. Lembrando que o AutonomousDB foi modelado utilizando o paradigma de orientação a objetos e, por isso, é recomendada a utilização de uma linguagem de programação orientada a objetos, que seja capaz de expressar de forma fidedigna o que foi modelado. Outros motivos para a escolha de Java foi a possibilidade de utilizar *frameworks* específicos para o desenvolvimento de Sistemas Multi-Agentes, além de ter um bom suporte e documentação para o esclarecimento de dúvidas. A se-

guir, serão apresentados os detalhes de implementação da Camada de Agentes e da Base de Conhecimento.

### 5.2.1 Camada de Agentes

Como explicado em sua arquitetura (ver Seção 4.2.1), o AutonomousDB possui uma Camada de Agentes composta por um SMA. Para dar suporte ao desenvolvimento de agentes inteligentes e o gerenciamento da comunicação entre eles, foi usado o *Framework* JADE (*Java Agent Development Framework*) [Jade, 2008]. JADE oferece uma plataforma específica para ser utilizada no desenvolvimento de sistemas multi-agentes e é totalmente codificado em Java. Oferece também um conjunto de ferramentas gráficas que dão suporte a *debugging* em todas as fases de desenvolvimento, além de darem suporte ao gerenciamento de agentes inteligentes em ambientes distribuídos e heterogêneos, recurso que é completamente alinhado com os objetivos do AutonomousDB.

JADE simplificou o desenvolvimento do SMA (Camada de Agentes), garantindo um padrão de interoperabilidade entre os agentes implementados (sensor, coordenador e atuadores) através de um conjunto de agentes de serviços de sistema. Um dos principais serviços utilizados pelos agentes implementados é o chamado serviço de *Páginas Amarelas* que, no JADE, é fornecido por um agente específico chamado *Directory Facilitator* (DF). Através dele, cada agente implementado pode cadastrar os serviços que oferece e procurar por serviços oferecidos por outros agentes. Uma vez que um agente implementado consulta o DF e sabe que um determinado serviço está sendo oferecido e por quem ele é fornecido, ele pode interagir com esse agente fornecedor através da troca de mensagens. As mensagens trocadas entre os agentes são capazes de trafegar objetos serializados de um agente para outro. No caso do AutonomousDB eles são instâncias da classe *DadosEventoNovo* e representam os dados dos eventos a serem executados. Dessa forma, o Agente Sensor consegue encaminhar os eventos para o Coordenador que os distribui entre os Agentes Atuadores. Os elementos básicos de um agente implementado utilizando JADE são: os comportamentos (*behaviors*), que implementam as ações (sejam elas atômicas ou compostas) executadas pelo agente; e a fila de mensagens, que gerencia individualmente o envio e o recebimento das mensagens de cada agente. A execução dos agentes é controlada por um escalonador, que controla de forma concorrente a execução dos comportamentos dos agentes (*threads*).

Para a escolha de JADE, outras plataformas de apoio ao desenvolvimento de sistemas multi-agentes também foram estudadas, a exemplo da: *JAgent (Generic multi-agents system test and development platform)* [3], *SACI (Simple Agent Communication Infrastructure)* [4] e a *Voyager* [5], mas nenhuma delas apresentou recursos suficientes para satisfazer de forma completa às necessidades de implementação do AutonomousDB.

### 5.2.2 Base de Conhecimento

Para a Base de Conhecimento do AutonomousDB foi utilizado o SGBD Oracle 10g Express Edition. Ele fornece os mesmos recursos do Oracle 10g com algumas limitações: i) armazenamento limitado em 4 (quatro) *gigabytes*; ii) limite de 1 (um) processador na máquina que o executa, ou seja, se o servidor possuir mais de 1 processador o excedente será ignorado pelo SGBD; iii) limite máximo de alocação de memória até 1 (um) *gigabyte*. Mesmo que o servidor tenha mais memória, o SGBD ignora o excedente. Nenhuma dessas limitações comprometem o correto funcionamento do AutonomousDB. Pelo contrário, já são recursos computacionais mais que suficientes. A maior motivação para a escolha do Oracle 10g Express Edition foi o fato dele fornecer todos os recursos do já conceituado SGBD Oracle 10g, sua administração é toda feita via *browser* através de uma interface bem intuitiva, além de ser gratuito. Apesar das limitações, ele satisfaz às necessidades do AutonomousDB. Se existir a necessidade de substituir o SGBD da Base de Conhecimento (seja por estouro da capacidade, questões de desempenho ou políticas) por outro, o AutonomousDB foi projetado para permitir essa substituição de forma fácil. No caso do novo SGBD local não ser da plataforma Oracle (ex. MySql, PostGres ou DB2), será necessário apenas uma pequena manutenção no componente *Controlador* para adaptar as consultas ao novo SGBD. Por motivos de segurança e manutenção da consistência, é fortemente aconselhável que exista apenas 1 (uma) instância do SGBD local para quantas forem as instâncias do AutonomousDB executando. Porém, nada impede que cada instância do AutonomousDB tenha seu SGBD local distinto. Nesse último caso, o responsável pela configuração terá que assumir todo o papel de gerenciar os diversos SGBD locais e assumir todos os riscos.

[3] http://www.jagentframework.org
[4] http://www.lti.pcs.usp.br/saci
[5] http://www.recursionsw.com

É importante lembrar que a Base de Conhecimento armazena os eventos fornecidos pelos DBA, agendamentos de execução, informações para gerar o *log*, tais como: dados funcionais dos agentes inteligentes e do ambiente. Isso facilita mudanças de qualquer natureza, seja no ambiente ou nos agentes, sendo apenas necessário atualizar a Base de Conhecimento e da próxima vez que o sistema inicializar, ele já assumirá as mudanças.

## 5.3 Experimento Realizado

A equipe de DBA do SERPRO Recife (ACDBA) que dá suporte ao SCC é composta por 11 profissionais, porém para o experimento foram selecionados 6 (seis). O critério de seleção foi os DBA que estão mais ligados às atividades de evolução dos esquemas dos módulos do SCC. Por motivo de segurança das demandas em desenvolvimento que utilizam os esquemas do banco *D0815*, uma cópia dele (somente os esquemas, sem dados) foi realizada e transferida para outra máquina. Essa cópia é a que foi utilizada efetivamente para o experimento. Foram utilizadas 4 (quatro) máquinas: 1 (uma) executando o SGBD (Oracle 10g) com a cópia do *D0815* (banco alvo), 1 (uma) executando o SGBD (Oracle 10g Express Edition) com a Base de Conhecimento e 2 (duas) executando instâncias do AutonomousDB. O sistema operacional de todas as máquinas utilizadas foi o Windows XP Professional [Microsoft, 2008], o ambiente de desenvolvimento foi o Eclipse 3.2.2 [Eclipse, 2008], a *Java Virtual Machine (JVM)* foi a 1.0.6 [Sun, 2008b] e o driver de conexão com os SGBD foi o *Java Database Connectivity (JDBC)* [Sun, 2008a].

Na Base de Conhecimento foi criado o modelo de persistência do AutonomousDB, especificado no Capítulo 4. Em seguida, foram carregadas as tabelas de domínio: *BANCO*, com dados de *login* e senha dos usuários do *D0815* e suas dependências; *AGENTES*, com dados de inicialização dos agentes, serviço executado e ação específica; *TABELAS*, com dados relevantes sobre todas as tabelas do *D0815*; *TRIGGERS*, com dados das *triggers BIU* referentes às tabelas (para efeito de geração do código fonte das *triggers*), e *PROCEDURES* com dados sobre as *procedures* que compõem as *update log packages* (para efeito de geração do código fonte das *packages*).

Para localizar bancos de dados distribuídos entre várias instâncias do SGBD, o Oracle 10g utiliza um recurso chamado *TNS Names*. Ele mantém um arquivo de controle

(*tnsnames.ora*), no qual guarda todas as informações sobre a instância do Oracle 10g e seus respectivos bancos de dados, tais como: *host*, protocolo, porta e identificador do banco (sid). Com a criação de uma cópia do *D0815* executando em uma máquina, e a Base de Conhecimento em outra, uma cópia do arquivo *tnsnames.ora* original utilizado no ambiente do SERPRO foi feita e adaptada para o experimento com informações das novas máquinas.

Configurados os SGBD para a Base de Conhecimento e a cópia do *D0815*, com suas respectivas instâncias executando, foi solicitado a cada um dos 6 (seis) DBA escolhidos para criarem e executarem eventos (criação, deleção, renomeação e mudança de tipo de atributos) que simulassem uma demanda de evolução de esquemas para os módulos do SCC que eles são responsáveis por manter. Os DBA também realizaram agendamentos para execução de eventos, edição de eventos, reaplicação de eventos em erro, consultaram *logs* de execução, excluíram eventos adicionados e revisaram códigos-fonte de *triggers BIU* e *update log packages* gerados. Dessa forma, cobriram todos os requisitos implementados nessa primeira versão do AutonomousDB. Todos os eventos fornecidos como entrada para o sistema correspondem a tarefas que seriam repetidas manualmente pelo DBA o número de vezes que o esquema alvo da mudança estivesse replicado no ambiente. Usando o AutonomousDB, o DBA só precisa informar a operação que deseja realizar, alguns dados de entrada (ex. nome da tabela alvo, atributo) e o esquema alvo da mudança. Todo o processo de propagação das atualizações é feito automaticamente (para as tabelas alvo, tabelas de histórico, *triggers BIU* e *update log packages*). Na Seção 5.5, vamos analisar os resultados deste experimento.

## 5.4 Resultados Obtidos

Após a execução do experimento com os DBA do SERPRO, vamos agora analisar e descrever os principais resultados. Primeiramente, foi analisado a Base de Conhecimento e constatado que todos os dados relativos aos eventos fornecidos como entrada pelos DBA foram corretamente armazenados. É importante enfatizar que 1 (um) evento criado pelo DBA através da GUI do sistema não corresponde necessariamente a 1 (um) evento armazenado na Base de Conhecimento para ser executado. Por exemplo, se algum DBA criar 1 (um) evento para a evolução do esquema CDCR (módulo Núcleo) no

banco *D0815*, o AutonomousDB irá verificar as replicações, armazenar e executar 42 eventos. Isso porque existem 7 (sete) módulos do SCC com 6 (seis) usuários disponíveis cada um, que tem o esquema CDCR replicado (ver Quadro 5.1). Já se esse evento fosse criado para o esquema SNEG, o AutonomousDB armazenaria e executaria apenas 6 (seis) eventos, pois o esquema SNEG só é utilizado pelo seu módulo correspondente (Saldos Negativos) e replicado em 6 (seis) usuários.

Após a verificação do correto armazenamento dos eventos na Base de Conhecimento, o próximo passo foi verificar a execução deles no banco *D0815*. É importante destacar que a correta execução de um evento não consiste apenas em criar, deletar, renomear ou mudar o tipo de atributos para uma tabela de negócio de um determinado esquema. Além disso, essa mesma ação tem que ser executada para as tabelas de histórico dos atributos; e os códigos fonte das *triggers BIU* e *update log packages* serem gerados. Todas essas operações realizadas corretamente caracterizam um evento executado com sucesso.

A Tabela 5.1 mostra o número de eventos criados por cada DBA que participou do experimento, os esquemas, o número de eventos efetivamente armazenados (depois da verificação das dependências) e executados. Por ela podemos ver que, se os DBA fossem realizar manualmente, todas as tarefas necessárias para cobrir a execução dos eventos deste experimento, eles teriam que executar as seguintes ações 1260 vezes: i) logar; ii) aplicar o comando DDL correspondente; iii) atualizar a *trigger BIU* e a *update log package* e iv) controlar o que foi atualizado ou não. Com isso, podemos concluir que para uma grande quantidade de ações e usuários de banco de dados com esquemas replicados, realizar essas tarefas manualmente é inteiramente improdutivo. Ainda mais quando se trata de tarefas que podem ser automatizadas.

No SERPRO, cada módulo do SCC tem um DBA específico para gerenciar sua evolução de esquemas. Por esse motivo, para o experimento, cada DBA ficou responsável por criar eventos no esquema o qual gerencia no seu dia-a-dia de trabalho. Segundo a Tabela 5.1, o DBA 1 (um) criou 10 (dez) eventos para serem executados no esquema PRRC. Como este esquema está replicado em 48 usuários (todos), o AutonomousDB após verificar as replicações, armazenou 480 eventos para execução. No caso do DBA 2 (dois), foram criados 10 (dez) eventos para o CDCR, que está replicado em 42 usuários, totalizando 420 eventos para execução. Os outros casos seguem o mesmo raciocínio.

Tabela 5.1 Resultados dos eventos criados e executados pelos DBA no experimento.

| DBA | Esquema | Eventos Criados | Eventos Executados |
|---|---|---|---|
| 1 | PRRC | 10 | 480 |
| 2 | CDCR | 10 | 420 |
| 3 | CCON | 8 | 288 |
| 4 | SNEG | 6 | 36 |
| 5 | PGIM | 4 | 24 |
| 6 | DIPI | 2 | 12 |
| TOTAL | 6 esquemas | 40 | 1260 |

Foram criados mais eventos para os esquemas PRRC, CDCR e CCON, porque são esquemas pertencentes a módulos-chave para o SCC e que têm o maior número de replicações. O restante dos esquemas só estão replicados para os seus 6 (seis) usuários correspondentes (ver Quadro 5.1).

## 5.5 Avaliação dos Resultados

Para garantir que os eventos fornecidos como entrada pelos DBA fossem executados com sucesso, os esquemas de todos os usuários afetados pelas modificações foram verificados um a um (tabelas de negócio e histórico). Junto com os esquemas, os códigos fonte das *triggers BIU* e *update log packages* gerados também foram revisados junto com os DBA. O agendamento de eventos também foi testado: 8 (oito) eventos foram agendados para horários específicos e executaram com sucesso. Para validar o comportamento do AutonomousDB em caso de erro, 5 (cinco) eventos extras foram criados para serem executados em tabelas que não existiam nos esquemas do *D0815*. O sistema abortou corretamente a execução para esses 5 (cinco) casos, registrou o *log* e a tabela de erros, colocando esses eventos com o *status* de "não concluído". Corrigidos os 5 (cinco) eventos em erro, eles foram reaplicados com sucesso. Ao final de toda essa análise, o *log* de execução gerado pelo AutonomousDB foi conferido. Com isso verificamos que todo o processo de evolução de esquemas e propagação de atualizações foi executado com sucesso, com os requisitos implementados para a ferramenta validados pelos DBA.

Toda a equipe de DBA (11 profissionais) que dá suporte ao SCC no SERPRO é

especialista no SGBD Oracle 10g. Porém, existem 5 (cinco) que são especialistas em MySQL, 4 (quatro) em SQL Server e 1 (um) em ADABAS, e há DBA com mais de uma especialidade. A média de experiência da equipe como especialista no SGBD Oracle 10g é de 6 (seis) anos. Para avaliar o AutonomousDB, um questionário contendo 5 (cinco) questões sobre as potenciais contribuições da ferramenta para o processo de trabalho dos DBA foi aplicado com os participantes do experimento (6 profissionais). O questionário foi desenvolvido com base nos objetivos gerais do AutonomousDB, para que fosse possível identificar se eles foram atingidos e se a ferramenta de fato é efetiva no trabalho dos DBA. As questões que compunham o questionário foram:

1. A ferramenta facilita seu dia a dia de trabalho? Em que aspectos?
2. Você acha que ganhou tempo livre com a ferramenta?
3. A ferramenta é fácil de usar?
4. Você usaria a ferramenta no seu trabalho?
5. Que sugestões você daria para melhorar a ferramenta?

O questionário foi respondido por cada um dos DBA e enviado por e-mail. Analisando as respostas, vamos descrever de uma forma geral a opinião deles.

Os DBA acharam que, sem dúvida, a ferramenta facilita o dia-a-dia de trabalho deles, principalmente na questão da considerável redução de trabalho repetitivo para evoluir os esquemas, atualizar as *triggers* e *update log packages*. Como todo o processo é feito de forma automática, existe uma maior segurança deles em relação à garantia da consistência dos vários esquemas, reduzindo a probabilidade de falhas humanas, economizando tempo e mão-de-obra. Foi relatado também que o fato de fazer a mesma tarefa várias vezes e com bastante atenção para não esquecer nenhum detalhe torna-se um trabalho muito cansativo. Alguns DBA passavam dias para completar demandas de evolução maiores e, com a ferramenta, ganham um tempo considerável para se dedicarem a atividades mais importantes que operacionais, tais como atividades de análise e projeto de banco de dados.

Os DBA também relataram o fato da ferramenta ser fácil de usar (boa usabilidade) e sua praticidade. A geração de *log* de processamento e controle de eventos em erro foram funcionalidades bastante citadas, pois todo esse controle é hoje feito informalmente,

causando muitas falhas de comunicação e conseqüentes problemas de inconsistências nos esquemas. O *log* de processamento serve como um documento formal de referência que determinado esquema foi evoluído corretamente ou não, já que nele são gravadas informações sobre as operações de evolução realizadas, tais como: data, hora, que ação foi executada, em que banco/usuário e se a ação foi executada com sucesso ou falha. O controle de erros permite atacar problemas pontuais e descobrir falhas antes não percebidas. Outros pontos fortes da ferramenta citados foram: autonomia, agendamento de eventos, habilidade de trabalhar em ambientes distribuídos, gerar código de *triggers* e *update log packages*. Todos os pontos positivos relatados pelos DBA foram objetivos propostos desde a especificação do AutonomousDB, mostrando que foram atingidos. Os DBA, de forma unânime, também relataram que com certeza usariam a ferramenta se ela fosse implantada.

As críticas sobre a ferramenta, foram basicamente em relação à funcionalidade de gerar *triggers BIU*. O AutonomousDB gera as *triggers* seguindo um padrão definido pelo SERPRO, levando em conta os novos atributos das tabelas após a evolução. O fato do AutonomousDB usar um padrão, pode forçar o DBA a ter que fazer customizações nas *triggers* novas, antes já realizadas nas antigas, causando um certo trabalho repetitivo que o gerador de *triggers* não prevê. Por esse motivo, o AutonomosDB não aplica automaticamente, mas gera o código fonte para revisão do DBA. A sugestão seria fazer o AutonomousDB apenas complementar o código fonte das *triggers* já existentes no banco com os trechos referentes à modificação e não gerá-lo novamente seguindo o padrão. Dessa forma não existiria necessidade do DBA ter o trabalho de comparar a *trigger* antiga com a nova para verificar se existem customizações. Isso é discutível, pois gerando o código baseado num padrão, o sistema consegue garantir a consistência das *triggers*. No caso de complementar o código existente, se as *triggers* estiverem inconsistentes, elas continuarão inconsistentes. Então, gerar as *triggers* ou complementá-las é uma escolha entre garantir a consistência delas ou diminuir o trabalho de verificar suas customizações.

O líder da equipe de DBA deu a sugestão de implementação de uma nova funcionalidade para o AutonomousDB. Essa funcionalidade permitiria a ferramenta executar *scripts* em usuários específicos, tornando-a ainda mais aderente ao processo de trabalho da equipe. O fato de ser possível aplicar *scripts* também permitiria ao AutonomousDB

realizar a evolução das instâncias de dados relativas aos esquemas evoluídos, pois nessa primeira versão, ele não trata a propagação da evolução de esquemas para as instâncias. O líder relatou também que se essa funcionalidade for implementada, o AutonomousDB irá satisfazer à maioria das demandas de evolução de esquemas para o SCC nos ambiente de desenvolvimento e testes do SERPRO.

Devido à sua aplicabilidade, o AutonomousDB está sendo testado e estendido para ser utilizado como ferramenta formal de apoio à evolução de esquemas no SERPRO. Os testes estão sendo feitos para o ambiente de desenvolvimento e testes, para em um futuro próximo contemplar também os ambientes de homologação e treinamento. Com isso, comprovamos a efetividade do AutonomousDB na solução de um problema real de evolução de esquemas, atingindo os objetivos propostos inicialmente e complementando-os com propostas de extensão e melhorias.

## 5.6 Experimento Adicional

O experimento realizado no SERPRO (descrito na Seção 5.3) foi executado em um ambiente de banco de dados que utiliza apenas a plataforma Oracle 10g. Isso porque o Oracle 10g é o SGBD oficial adotado pelo SERPRO por meio de licitação. Para testar a capacidade do AutonomousDB de operar em ambientes com SGBD heterogêneos, um experimento adicional foi realizado. Nesse experimento adicional, foi utilizado além do Oracle 10g, o MySQL [MySQL, 2008]. Como o esquema físico depende do SGBD, o custo de estender o gerador de *triggers* e o de *packages* para gerarem código fonte também para o MySQL, seria alto. Por esse motivo, no caso do MySQL, apenas as funções de adição, deleção, renomeação e mudança de tipo de atributos foram implementadas.

No MySQL foi criado um banco de dados e nele replicado o esquema CDCR (ver Quadro 5.1), que também está no banco *D0815* da plataforma Oracle 10g. Feito isso, 5 (cinco) eventos foram criados para realizarem modificações no esquema CDCR. Todos eles foram executados com sucesso, evoluindo o esquema em questão nas duas plataformas. Lembrando que para o MySQL não foram implementadas as funções de geração de *triggers* e *packages*. Para testar o tratamento de erros, o esquema CDCR foi apagado propositalmente do MySQL e, logo após, 2 (dois) eventos foram criados para este esquema. O AutonomousDB evoluiu o esquema CDCR presente na plataforma Oracle

10g e levantou erro para o MySQL, suspendendo os eventos (colocando-os com o *status* de pendente na Base de Conhecimento) e registrando *log*. Os detalhes deste experimento, bem como resultados, estão publicados no trabalho [Moraes et al., 2009]. Com isso, foi testada a efetividade do AutonomousDB em ambientes que utilizam SGBD heterogêneos.

## 5.7 Conclusão

Neste capítulo, foi apresentado o ambiente de banco de dados do SERPRO e escolhido um banco de desenvolvimento, no qual uma implementação do AutonomousDB foi aplicada. Foi explicado também em detalhes a implementação da ferramenta, justificando o uso de tecnologias. Terminada a implementação, foi detalhado o experimento realizado e seus principais resultados obtidos. No estudo dos sistemas existentes apresentados no Capítulo 3, não encontramos outras soluções para o problema de propagação automática de esquemas utilizando sistemas multi-agentes em ambientes distribuídos e heterogêneos. Além disso, a solução proposta resolve um problema existente em um ambiente real de forma satisfatória, de acordo com os DBA consultados. No Capítulo 6, vamos concluir o trabalho, apresentando as contribuições e potenciais trabalhos futuros.

# CAPÍTULO 6
# Conclusões

Este último capítulo apresenta uma síntese do trabalho desenvolvido e aponta suas principais contribuições. Também relatamos as possibilidades de pesquisa que tenham como referência o AutonomousDB.

## 6.1 Resumo do Trabalho Desenvolvido

O objetivo principal do trabalho foi desenvolver uma solução baseada em sistemas multi-agentes (SMA) capaz de prover autonomia na tarefa de administração de banco de dados. Mais especificamente no que se diz respeito à evolução e propagação de mudanças/atualizações de esquemas replicados em ambientes distribuídos e heterogêneos. Inicialmente, abordamos os principais conceitos e características da Computação Autônoma, suas aplicações gerais e específicas ao contexto de SGBD. Explanamos também os principais conceitos sobre Evolução de Esquemas, apresentando sua contextualização e necessidade de tratamento de sua problemática. Também apresentamos os requisitos básicos a serem atendidos para uma evolução de esquemas satisfatória e a descrição das principais técnicas e estratégias utilizadas para realizá-la. Para uma completa formalização do conhecimento, fizemos um levantamento das principais iniciativas de apoio à tarefa de evolução de esquemas, mostrando o comparativo de algumas de ordem acadêmica e de mercado, em relação à solução proposta nesse trabalho.

Explanados os conceitos de Computação Autônoma e Evolução de Esquemas, propusemos a solução AutonomousDB, inspirada na combinação dos conceitos das duas áreas apresentadas para amenizar o problema da evolução de esquemas. Para demonstrar a efetividade da solução, implementamos um protótipo baseado no modelo de projeto resultante do seu processo de modelagem. Esse protótipo foi implementado com base no ambiente de banco de dados de desenvolvimento do SERPRO, onde existem es-

quemas replicados em vários usuários. Isso causa um sério problema de manutenção da consistência dos esquemas evoluídos e trabalho repetitivo por parte dos DBA. Aplicado o protótipo, 6 (seis) DBA do SERPRO que participaram do experimento, responderam a um questionário para validar o AutonomousDB. Os questionários foram analisados e deles tiradas conclusões para a apresentação dos principais resultados.

## 6.2 Contribuições

As principais contribuições deste trabalho foram:

1. Comprovação da efetividade de conceitos da Computação Autônoma aplicados a SGBD, satisfazendo a necessidade de controle da crescente complexidade desse tipo de sistema para atender a diversos contextos;
2. Criação de uma ferramenta flexível, baseada em agentes inteligentes, capaz de atuar em ambientes de banco de dados distribuídos e heterogêneos, atacando o problema da inconsistência de esquemas replicados. A solução foi projetada para ter um alto grau de modularidade, possibilitando a troca ou adição de novos componentes (agentes, Base de Conhecimento ou SGBD alvo);
3. Proposta de uma solução que usa tecnologias de código aberto e são multi-plataforma, além de poder ser aplicada a qualquer SGBD;
4. Definição de papéis e funcionalidades a serem desempenhados por cada agente inteligente e a forma de comunicação entre eles;
5. Definição da forma de coordenação dos agentes;
6. Estabelecimento de um modelo de persistência de informações que permite maior robustez à arquitetura da solução;
7. Controle automatizado de eventos de evolução de esquemas executados nos SGBD alvo. Isso tira a responsabilidade do DBA de fazer o controle dos esquemas que foram ou não evoluídos dentro de um ambiente, evitando inconsistências;
8. Construção de um protótipo baseado em um problema real, com resultados positivos comprovados. Esse protótipo é capaz de reduzir consideravelmente o trabalho de uma equipe de DBA na manutenção da consistência dos esquemas, *triggers* e *update log packages*. Isso libera os DBA para se dedicarem a tarefas mais importantes.

## 6.3 Trabalhos Futuros

Apesar desse trabalho ter atingido seus objetivos iniciais, podemos identificar a necessidade de um período maior de avaliação e refinamento do AutonomousDB para que a ferramenta possa ter uma primeira versão em produção. O contexto da problemática de evolução de esquemas é muito amplo e tende a aumentar cada vez mais com a crescente complexidade dos sistemas. Para termos uma idéia, o AutonomousDB trata de apenas uma parte dessa problemática e nele existem muitas situações específicas que ainda podem ser tratadas. A seguir, listamos alguns melhoramentos e trabalhos possíveis de serem feitos em cima do que já existe do AutonomousDB.

1. Melhoramento e extensão dos geradores de código fonte das *triggers* e *update log packages*. Na versão atual do AutonomousDB só é gerado código fonte para a plataforma Oracle. Os geradores poderiam ser estendidos para gerar código para outras plataformas como: *MySQL* [MySQL, 2008], *PostGres* [PostGres, 2008] ou *DB2* [IBM, 2008]. Eles também poderiam ser melhorados no sentido de poderem carregar *templates* com informações (ex: sintaxe de comandos DDL) sobre as várias plataformas que eles poderiam gerar código (parametrização). Esses *templates* poderiam estar descritos em arquivos *XML*, que seriam carregados ou descarregados dos geradores, correspondendo à capacidade de dar suporte ou não à geração de código fonte para uma determinada plataforma;
2. Implementação de novas funcionalidades. O fato do AutonomousDB ser baseado numa arquitetura flexível, baseada em agentes, dá a ele a capacidade de incorporar mais facilmente novas funcionalidades, como por exemplo a de executar *scripts*;
3. Estender o Agente Sensor para de tempos em tempos tentar executar novamente os eventos com *status* de pendente na Base de Conhecimento (melhor esforço);
4. Aumento do escopo de atuação. Essa primeira versão do AutonomousDB não propaga as modificações dos esquemas para as instâncias de dados correspondentes. Uma extensão nesse sentido, tornaria a solução bem mais completa;
5. O raciocínio e comunicação entre os agentes poderia ser melhor refinada, dessa forma eles seriam capazes de executar tarefas mais complexas dentro do ambiente de banco de dados;

6. Especificação de um modelo para armazenar, classificar e interpretar as regras de negócio e políticas organizacionais; e
7. Especificação de um *Data Warehouse* para análise de desempenho da ferramenta (com elaboração de critérios de qualidade e indicadores de desempenho). Isso permitiria uma avaliação constante do sistema.

Dessa forma, o AutonomousDB poderá ser continuado por desenvolvedores em geral. Eles podem, por exemplo, desenvolver novos agentes para serem acoplados ao sistema e realizarem novas ações. Esses novos agentes podem tratar situações mais específicas. A arquitetura flexível do AutonomousDB permite esse tipo de extensão sem maiores custos.

# Referências Bibliográficas

[Al-Jadir, 1997] Al-Jadir, L. (1997). "Evolution-Oriented Database Systems". Thesis 1997, Geneva University, Switzerland.

[Al-Jadir & Leonard, 1998] Al-Jadir, L. & Leonard, M. (1998). "Multiobjects to Ease Schema Evolution in an OODBMS". In Int. Conf. on Conceptual Modeling (ER 98), Vol. 1507, p. 316-333, Lecture Notes in Computer Science, 1999.

[Andany et al., 1991] Andany, J. C., Conard, M. L. & Palisser, C. (1991). "Management Of Schema Evolution In Databases". Swiss Research Fundation, Rebirth project, Proceedings of the 17th International Conference, Barcelona September, 1991.

[Aumüller & Rahm, 2006] Aumüller, D. & Rahm, E. (2006). "Caravela: Semantic Content Management with Automatic Categorization". Universidade de Leipzig, 2006. Disponível em: http://se-pubs.dbs.uni-leipzig.de.

[Autoadmin, 2007] Autoadmin (2007). "AutoAdmin: Self-Tuning and Self-Administering Databases". Disponível em: http://www.research.microsoft.com/dmx/autoadmin/ - Último acesso em 15/07/2008.

[Autonomic Computing, 2007] Autonomic Computing (2007). Disponível em http://www.autonomiccomputing.org/. Último acesso em: 02/07/2008.

[Banerjee et al., 1987a] Banerjee, J., Chouand, H. T., Garzaa, J. F., Kim, W., Woelk, D., Ballou, N. & Kim, H. J. (1987a). "Data model issues for object-oriented applications". In ACM Transactions on Office Information Systems, Vol. 5, No. 1, p. 3-26, 1987.

[Banerjee et al., 1987b] Banerjee, J., Kim, W. & Korth, H. F. (1987b). “Semantics and Implementation of Schema Evolution in Object-Oriented Databases”. Proc. ACM SIGMOD, San Francisco, 1987.

[Benatallah, 1999] Benatallah, B. (1999). “A Unified Framework for Supporting Dynamic Schema Evolution in Object Databases”. In Int. Conf. on Conceptual Modeling (ER 99), Vol. 1728, p. 16-30, Lecture Notes in Computer Science, 1999.

[Bjornerstedt & Hulten, 1989] Bjornerstedt, A. & Hulten, C. (1989). “Version Control in a Object-Oriented Architecture”. in Object-Oriented Concepts, Databases and Applications, Addison-Wesley Ed., p 451-485, 1989.

[Blaschka, 2000] Blaschka, M. (2000). “A Framework for Schema Evolution in Multidimensional Databases”. Ph.D. Thesis - Technische Universität München, Germany.

[Bounif, 2004] Bounif, H. (2004). “Predictive Approach for Database Schema Evolution”. Database Laboratory, Swiss Federal Institute of Technology, Switzerland.

[Butterworth et al., 1991] Butterworth, P., Otis, A. & Stein, J. (1991). “The Gemstone Object Management”. In Comm. of the ACM, Vol. 34, No. 10, p. 64-77, 1991.

[Camolesi & Traina, 1996] Camolesi, L. & Traina, C. (1996). “Evolução de Esquemas de Dados: Um Panorama Amplo de Aspectos Técnicos e Gerenciais”. XI Simpósio Brasileiro de Bancos de Dados (SBBD), Sociedade Brasileira de Computação (SBC), Universidade Federal de São Carlos (UFSCar), São Carlos, Outubro 1996, p. 1-19.

[Carneiro et al., 2004] Carneiro, A., Passos, R., Belian, R., Costa, T., Tedesco, P. & Salgado, A. C. (2004). “DBSitter: An Intelligent Tool for Database Administration”. Em DEXA, 15th International Conference and Workshop on Database and Expert Systems Applications.

[Chikovsky & Cross, 1990] Chikovsky, E. J. & Cross, J. (1990). “Reverse Engineering and Design Recovery: A Taxonomy”. IEEE Software, Vol. 7, No. 1, p. 13-18, Janeiro 1990.

[Coleman, 2007] Coleman, T. (2007). Evolution of Online Schema Change to Database Definition on Demand. DB2utor. Disponível em:

http://ibmsystemsmag.blogs.com/db2utor/2007/09/evolution-of-on.html - Último acesso em: 24/09/2008.

[Constantinescu, 2003] Constantinescu, Z. (2003). “Towards an Autonomic Distributed Computing System”. in Em Proceedings of the 14th International Workhop on Database and Expert Systems Applications (DEXA OS).

[de Albuquerque, 2007] de Albuquerque, A. E. X. C. (2007). “Um Mecanismo de Coordenação para o Framework xaADB”. Trabalho de Graduação - Universidade Federal de Pernambuco, Centro de Informática (CIn-UFPE).

[de Matos Galante, 2003] de Matos Galante, R. (2003). “Modelo Temporal de Versionamento com Suporte à Evolução de Esquemas”. Tese de Doutorado - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[Dominguez et al., 2008] Dominguez, E., Lloret, J., Rubio, A. L. & Zapata, M. A. (2008). “MeDEA: A database evolution architecture with traceability”. In Proceedings of Data Knowledge Engineering, p.419-441, 2008.

[Eclipse, 2008] Eclipse (2008). “Eclipse Official Website”. Disponível em: http://www.eclipse.org. Último acesso em: 12/10/2008.

[Elnaffar et al., 2003] Elnaffar, S., Powley, W., Benoit, D. & Martin, P. (2003). “Today’s DBMSs: How Autonomic are they?”. Technical Report 2003-469, School of Computing, Queen’s University, Kingston, Ontario, Canada.

[Englebert, 2004] Englebert, V. (2004). “Voyager 2 reference manual”. Version 7, Release 0, June 2004.

[Englebert & Hainaut, 1999] Englebert, V. & Hainaut, J.-L. (1999). “DB-main: a next generation meta-case”. Journal Information System, Vol. 24, No. 2, p. 99-112, 1999.

[Ferrandina et al., 1995] Ferrandina, F., Meyer, T., Zicari, R., Ferran, G. & Madec, J. (1995). “Schema and Database Evolution in the O2 Object Database System”. In International Conference on Very Large Databases,1995.

[Fuad & Oudshoorn, 2006] Fuad, M. M. & Oudshoorn, M. J. (2006). "An Autonomic Architecture for Legacy Systems". in Proceedings of the Third IEEE International Workshop on Engineering of Autonomic and Autonomous Systems, EASE 2006.

[Ganek & Corbi, 2003] Ganek, A. G. & Corbi, T. A. (2003). "The drawing of the autonomic computing era". Em IBM Systems Journal, Vol. 42, No. 1.

[GemStone, 2003] GemStone (2003). "GemStone Documentation - Release 6.0.1". GemStone System.

[Gomez & Olive, 2002] Gomez, C. & Olive, A. (2002). "Evolving Partitions in Conceptual Schemas in the UML". CAiSE 2002, Toronto, Ontario, Canada.

[Gomez & Olive, 2003] Gomez, C. & Olive, A. (2003). "Evolving Derived Entity Types in Conceptual Schemas in the UML". In the 9th International Conference, OOIS 2003:3-45, Geneva, Switzerland.

[Hainaut et al., 1994] Hainaut, J.-L., Englebert, V., Henrard, J., Hick, J.-M. & Roland, D. (1994). "Database Evolution: the DB-Main Approach". In Proc. of the 13th International Conference on ER Approach, p. 112-131, 1994.

[Henrard et al., 1996] Henrard, J., Englebert, V., Hick, J. M., Roland, D. & Hainaut, J. L. (1996). "DB-MAIN: un atelier d'ingénierie de bases de données". In Ingénierie des Systèmes d" Information, Vol. 4, No. 1, pp. 87-116, 1996.

[Hick, 2001] Hick, J. M. (2001). "Evolution of relational database applications: Methods and Tools". University of Namur, Computer Sciences Department, PhD Thesis, 2001.

[Hick & Hainaut, 2002] Hick, J. M. & Hainaut, J. L. (2002). "Strategy for Database Application Evolution: the DB-MAIN Approach". University of Namur, Computer Sciences Department, 2002.

[Horn, 2001] Horn, P. (2001). "Autonomic Computing: IBM's Perspective on The State of Information Technology". IBM Corporation. Disponível em: http://www.research.ibm.com/autonomic/ - Último acesso em: 04/07/2008.

[IBM, 2008] IBM (2008). "DB2 Product Family - Official Website". Disponível em: http://www-01.ibm.com/software/data/db2. Último acesso em: 12/10/2008.

[IBM Autonomic Blueprint, 2006] IBM Autonomic Blueprint (2006). "An architectural blueprint for autonomic computing". Quarta Edição, disponível em http://www.03.ibm.com/autonomic/pdfs - Último acesso em: 03/07/2008.

[Itasca, 1995] Itasca (1995). "ITASCA Distributed Object Database Management System - Technical Summary for Release 2.3.5". Itasca Systems.

[Jade, 2008] Jade (2008). "Java Agent DEvelopment Framework". Disponível em: http://jade.cselt.it. Último acesso em: 12/10/2008.

[Jasmine, 2003] Jasmine (2003). "The Jasmine Documentation". Computer Associates International.

[Java, 2008] Java (2008). "The Source for Java Technology". Disponível em: http://java.sun.com. Último acesso em: 24/09/2008.

[Jensen et al., 1998] Jensen, C. S., Dyreson, C. E., Bohlen, M. H., Clifford, J., Elmasri, R., Gadia, S. K., Grandi, F., Hayes, P. J., Jajodia, S., Kafer, W., Kline, N., Lorentzos, N. A., Mitsopoulos, Y. G., Montanari, A., Nonen, D. A., Peressi, E., Pernici, B., Roddick, J. F., Sarda, N. L., Scalas, M. R., Segev, A., Snodgrass, R. T., Soo, M. D., Tansel, A. U., Tiberio, P. & Wiederhold, G. (1998). "The Consensus Glossary of Temporal Database Concepts".

[Kephart & Chess, 2003] Kephart, J. O. & Chess, D. M. (2003). "The Vision of Autonomic Computing". IEEE Computer Society, Vol. 36, issue 1, jan 2003, p. 41-50.

[Kim & Chou, 1998] Kim, W. & Chou, H. T. (1998). "Versions of Schema for Object Oriented Databases". Proc. VLDB 88, Los Angeles, 1988.

[Koehler et al., 2003] Koehler, J., Giblin, C., Gantenbein, D. & Hauser, R. (2003). "On Autonomic Computing Architectures". Em Research Report (Computer Science) RZ 3487 (99302), IBM Research (Zurich).

[Lakshmanan, 1993] Lakshmanan, L. (1993). “On the Logical Foundations of Schema Integration and Evolution in Heterogeneous Database Systems”. Intl. Conference on Deductive and Object-Oriented Databases - DOOD, p. 81-100, 1993.

[Lammari et al., 2003] Lammari, N., Akoka, J. & Wattiau, I. C. (2003). “Supporting Database Evolution: Using Ontologies Matching”. in 9th International Conference, OOIS 2003 Geneva, Switzerland, September 2003.

[Lautemann, 1997] Lautemann, S. E. (1997). “Schema Versions in Object-Oriented Database Systems”. In Int. Conf. on Database Systems for Advanced Applications (DASFAA), Vol. 6, p. 323-332, World Scientific, 1997.

[Lautemann, 1999] Lautemann, S. E. (1999). “Change Management with Roles”. In Int. Conf. on Database Systems for Advanced Applications (DASFAA), Vol. 6, p. 291-300, IEEE Computer Society, 1999.

[Lightstone et al., 2003] Lightstone, S., Schiefer, B., Zilio, D. & Kleeewein, J. (2003). “Autonomic Computing for Relational Databases: The Ten-Year Vision”. IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003), Banff AB.

[Lohman & Lightstone, 2002] Lohman, G. M. & Lightstone, S. S. (2002). “SMART: Making DB2 (More) Autonomic”. Em 28th VLDB Conference, Very Large Database Endowment.

[Maciel, 2007] Maciel, P. R. M. (2007). “DBSitter-AS: um Framework Orientado a Agentes para Construção de Componentes de Gerenciamento Autônomo para SGBD”. Dissertação de Mestrado - Universidade Federal de Pernambuco, Centro de Informática (CIn-UFPE).

[Marques, 2002] Marques, T. (2002). “Desafios para o Futuro”. Revista Pesquisa FAPESP. Disponível em: http://www.revistapesquisa.fapesp.br/ - Último acesso em: 06/07/2008.

[Microsoft, 2008] Microsoft (2008). “Edições do Windows XP”. Disponível em: http://www.microsoft.com/brasil/windowsxp. Último acesso em: 27/02/2008.

[Moraes et al., 2009] Moraes, A. J. C., Salgado, A. C. & Tedesco, P. C. A. R. (2009). “AutonomousDB: a Tool for Autonomic Propagation of Schema Updates in Heterogeneous Multi-Database Enviroments”. In 50th Int. Conf. on Autonomic and Autonomous Systems (ICAS 2009), Valencia, Spain, April 2009.

[MySQL, 2008] MySQL (2008). “MySQL Official Website”. Disponível em: http://www.mysql.com. Último acesso em: 12/10/2008.

[Nguyen & Rieu, 1989] Nguyen, G. & Rieu, D. (1989). “Schema Evolution in Object-Oriented Database Systems”. In Data Knowledge Engineering, Vol. 4, No. 1,

[Noy & Klein, 2002] Noy, N. F. & Klein, M. (2002). “Ontology Evolution: Not the Same as Schema Evolution”. Knowledge and Information Systems, 2002.

[O2, 1999] O2 (1999). “The O2 System - Release 5.0 Documentation”.

[Objectivity, 2001] Objectivity, I. (2001). “Schema Evolution in Objectivity/DB”. An Objectivity, Inc. White Paper.

[ObjectStore, 2003] ObjectStore (2003). “ObjectStore Release 6.1”. Object Design.

[Oracle, 2005] Oracle (2005). “Oracle Database 10g Release 2 Online Data Reorganization & Redefinition”. An Oracle White Paper, May 2005.

[Poet, 1997] Poet (1997). “POET 5.0 Documentation Set”. POET Software.

[PostGres, 2008] PostGres (2008). “PostGres Official Website”. Disponível em: http://www.postgresql.org.br. Último acesso em: 12/10/2008.

[Ra, 2004] Ra, Y. G. (2004). “Relational Schema Evolution for Program Independency”. Lecture notes in computer science, No. 3356, p. 273-281, 2004.

[Rahm & Bernstein, 2006] Rahm, E. & Bernstein, P. A. (2006). “An on-line bibliography on schema evolution”. ACM SIGMOD Record, v.35, n.4, p.30-31, December 2006.

[Ram & Shankaranarayanan, 2003] Ram, S. & Shankaranarayanan, G. (2003). “Research issues in database schema evolution: the road not taken”. Working Paper 2003-15.

[Ramanujam & Capretz, 2005] Ramanujam, S. & Capretz, M. A. M. (2005). "ADAM: A Multi-Agent System for Autonomous Database Administration and Maintenance". International Journal of Intelligent Information Technologies, Vol. 1, No. 3, p.14-33, Jul-Set 2005.

[Rashid, 2001] Rashid, A. (2001). "A Database Evolution Approach for Object-Oriented Databases". In Int. Conf. on Software Maintenance (ICSM 2001), p. 561-564, IEEE Computer Society, 2001.

[Roddick, 1992] Roddick, J. F. (1992). "Schema Evolution in Database Systems - An Annotaded Bibliography". Sigmond Record, Vol.21, No. 4, December, 1992.

[Roddick, 1995] Roddick, J. F. (1995). "A survey of schema versioning issues for database systems". Information and Software Technology, Vol. 37, No. 7, p. 383-393, 1995.

[Roland et al., 2000] Roland, D., Hainaut, J. L., Hick, J. M., Henrard, J. & Englebert, V. (2000). "Database engineering processes with DB-MAIN". In Proc. of the 8th European Conference on Information Systems (ECIS2000), 2000.

[Russell & Norvig, 2003] Russell, S. J. & Norvig, P. (2003). "Artificial Intelligence: A Modern Approach". 2nd edition. Prentice Hall.

[Ryan et al., 2001] Ryan, C., Annie, A., Aldo, D. & Laurie, W. (2001). "Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model". IEEE 5th International Symposium on Requirements Engineering.

[Saccol & Edelweiss, 2005] Saccol, D. B. & Edelweiss, N. (2005). "Evolução de Esquemas em Ambientes de Integração de Dados". Trabalho Individual - Universidade Federal do Rio Grande do Sul, Institudo de Informática, Porto Alegre.

[Sayão, 2003] Sayão, M. (2003). "Quando utilizar Sistemas Multi-Agentes?". PUC-RS Faculdade de Informática.

[Sjoberg, 1993] Sjoberg, D. (1993). "Quantifying Schema Evolution". Information and Software Technology, Vol. 35, No. 1, p. 35-44, January 1993.

[Sockut & Goldberg, 1979] Sockut, G. H. & Goldberg, R. P. (1979). “Database Reorganization - Principles and Practice”. ACM Computing Surveys, Vol. 11, No. 4, Dezembro, p. 371-395, 1979.

[Sun, 2008a] Sun (2008a). “Java SE Technologies - Database”. Disponível em: http://java.sun.com/javase/technologies/database. Último acesso em: 12/10/2008.

[Sun, 2008b] Sun (2008b). “Java SE 6”. Disponível em: http://java.sun.com/javase/6. Último acesso em: 12/10/2008.

[Versant, 1997] Versant (1997). “Versant Manuals for Release 5.0”. Versant Object Tecnology.

[Wattiau et al., 2002] Wattiau, I. C., Akoka, J. & Lammari, N. (2002). “A Framework for Database Evolution Management”. Laboratoire CEDRIC-CNAM et ESSEC.

[Wedemeijer, 2000] Wedemeijer, L. (2000). “Defining Metrics for Conceptual Schema Evolution”. in 9th Int. Workshop on Foundations of Models and Languages for Data and Objects, FoMLaDO/DEMM 2000 Dagstuhl Castle, Germany, 2000.

[Weiss, 1999] Weiss, G. (1999). “Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence”. The MIT Press.

[Zdonik, 1993] Zdonik, S. B. (1993). “Incremental Database Systems: Databases from the Ground Up”. ACM SIGMOD Record, Vol. 22, No. 2, pp. 408-412, 1993.

[Zdonik & Mitchell, 1991] Zdonik, S. B. & Mitchell, G. (1991). “ENCORE: an object-oriented approach to database modelling and querying”. In Data Engineering Journal, IEEE Computer Society Press, Vol. 14, No. 2, p. 53-57, 1987.

[Zicari, 1990] Zicari, R. (1990). “Incomplete Information in Object-Oriented Databases”. ACM SIGMOD Record, Vol. 19, No. 3, p. 5-16, Setembro 1990.