Um Framework de Regras Ativas para Sistemas de Gerência de Banco de Dados Distribuído

Sérgio da Costa Côrtes
scortes@inf.puc-rio.br

Carlos José Pereira de Lucena
lucena@inf.puc-rio.br

PUC-RioInf.MCC16/01 Junho, 2001

Abstract

Active database management systems make possible the development of advanced database applications, giving support to a semantics that reflects the application behavior through the definition of rules, which specify actions that are automatically triggered by the occurrence of certain events. The use of active rules in a distributed environment allows obtaining more expressiveness, although with a considerable increase of the system complexity. This work proposes a framework for constructing active distributed database management systems that applies the reuse technique (*framework*) in order to benefit by the specifications which must exist in all systems of this kind, at the same time that allows being configured/extended to specific operational environments, with different rule semantics.

Keywords: framework, framelet, database, active rule, active distributed database management system

Resumo

Sistemas de Gerência de Banco de Dados com Regras Ativas (SGBDRA) possibilitam o desenvolvimento de aplicações de bancos de dados não convencionais, dando suporte a uma semântica que reflete o comportamento com base em eventos. A utilização de regras ativas em um ambiente distribuído permite obter mais expressividade, embora com um considerável aumento da complexidade do sistema. Neste trabalho, é proposto um framework para construção de Sistemas de Gerência de Banco de Dados Distribuído com Regras Ativas (SGBDDRA) que permite o desenvolvimento de diferentes SGBDDRA, reaproveitando as funcionalidades que sejam comuns e permitindo implementações específicas para diversos ambientes operacionais, com diferentes semânticas de regra.

Palavras-chave: framework, framelet, banco de dados, regras ativas, sistema de gerência de banco de dados distribuído com regras ativas

1. Introdução

Nesta seção, é apresentada a motivação deste trabalho que consiste, fundamentalmente, na proposta de um framework para a construção de SGBDDRA (Sistemas de Gerência de Banco de Dados Distribuído com Regras Ativas). É discutido o contexto no qual este trabalho está inserido, bem como, é apresentada sua organização.

1.1. Contexto

Sistemas de Gerência de Banco de Dados (SGBD) possuem uma posição de destaque nas tecnologias atuais de sistemas de informação. Eles provêm confiança, eficiência, acesso, alta disponibilidade e mecanismos eficazes para armazenamento e manutenção de grandes volumes de informações. Nos últimos anos, pesquisas e trabalhos práticos têm se direcionado para a criação de bancos de dados voltados para aplicações bastante diferenciadas das convencionais, estendendo sua semântica, de forma que o próprio SGBD as suporte. Bancos de dados com regras ativas, temporais, espaciais e multimídia são alguns exemplos, resultados dessas novas aplicações.

1.2. Motivação

Sistemas de Gerência de Banco de Dados com Regras Ativas (SGBDRA) possibilitam o desenvolvimento de bancos de dados não convencionais, dando suporte a uma semântica que reflete o comportamento com base em eventos ou acontecimentos, em um determinado domínio do banco de dados. Um Banco de Dados Distribuído (BDD) é uma coleção de dados que pertencem logicamente ao mesmo sistema, mas está fisicamente espalhado (distribuído) pelos *sites* de uma rede de computador. A utilização de regras ativas em um ambiente distribuído permite obter mais expressividade, embora com um considerável aumento da complexidade do sistema.

Os SGBDRA podem diferir em função da forma como controlam e administram a execução das regras, o modo de detectar seus eventos, bem como quanto ao aspecto de distribuição de regras e dados.

A técnica de framework oferece a infra-estrutura adequada para a construção de SGBDRA, já que um framework é um (sub)sistema de software, de um domínio particular, que pode ser ajustado para aplicações individuais. Um framework consiste em uma grande estrutura que pode ser reutilizada, como um todo, para a construção de um novo sistema, o que representa, neste contexto, os diferentes SGBDRA.

1.3. Organização

O restante deste trabalho está organizado da seguinte forma:

Na seção 2, é apresentada uma breve revisão dos principais conceitos sobre SGBDRA e Sistemas de Gerência de Bancos de Dados Distribuídos (SGBDD), necessários para o entendimento da proposta desse trabalho.

Na seção 3, os conceitos de framework relevantes a esta proposta são discutidos.

Na seção 4, é proposto um framework para a construção de SGBDRA. Seus hot spots e frozens spots são apresentados e seu uso é exemplificado através de uma instanciação do framework proposto.

Finalmente, a seção 5 contém uma síntese da pesquisa e aponta alguns trabalhos futuros.

2. Sistemas de Gerência de Banco de Dados

SGBD são um conjunto de softwares altamente complexos e sofisticados que possibilitam, basicamente, os serviços de armazenamento, recuperação e gerência de dados. Existe uma grande diversidade de arquiteturas de SGBD, no entanto, é possível identificar um conjunto básico de funcionalidades que normalmente consta das mais diferentes arquiteturas. Nesta seção, será apresentada uma arquitetura genérica de SGBD, identificando seus componentes e os relacionamentos existentes entre estes. Também é discutida a questão do comportamento ativo em SGBD.

2.1. Componentes de um SGBD

Um SGBD é composto por diversos módulos (componentes) de softwares, cada qual com uma função operacional específica. Muitas das funcionalidades dos SGBD são suportadas por funções dos sistemas operacionais, tais como exclusividade de acesso a arquivos, prioridades de execução de processos, entre outras. Assim sendo, os SGBD devem ser construídos como camadas acima dos sistemas operacionais, visando utilizar esses serviços.

Os principais componentes de software de um SGBD estão apresentados na Figura 1. Este diagrama também mostra quais são as interfaces entre tais componentes e entre estes e alguns componentes externos ao SGBD, tais como os módulos de Consultas dos Usuários e Método de Acesso (este pertencente ao sistema operacional).

Os principais componentes de um SGDB são:

⇒ ***Processador de Consulta***: Transforma as consultas submetidas ao SGBD em uma série de instruções de baixo nível para o Gerente de Banco de Dados.

⇒ ***Gerente de Banco de Dados***: Recebe solicitações relativas à submissão de consultas e programas de aplicações e examina os esquemas externo e conceitual para determinar que registros conceituais são necessários para satisfazê-las. Este módulo chama o Gerente de Arquivos para atender às suas solicitações. Este módulo está descrito na Figura 2.

⇒ ***Gerente de Arquivos***: Manipula os arquivos para armazenamento e gerencia a alocação dos espaços em disco. Estabelece e mantém uma lista com as estruturas e índices definidos no esquema interno do banco de dados. Se o sistema utiliza arquivos *hashed*, ele chama funções de *hashing* para gerar os endereços dos arquivos. Entretanto, o Gerente de Arquivos não gerencia diretamente os *inputs* e *outputs* físicos dos dados. Ele necessita de um método de acesso apropriado, o qual tanto lê quanto grava dados no *buffer* do sistema. Maiores detalhes de métodos de acessos podem ser encontrados em [EN99].

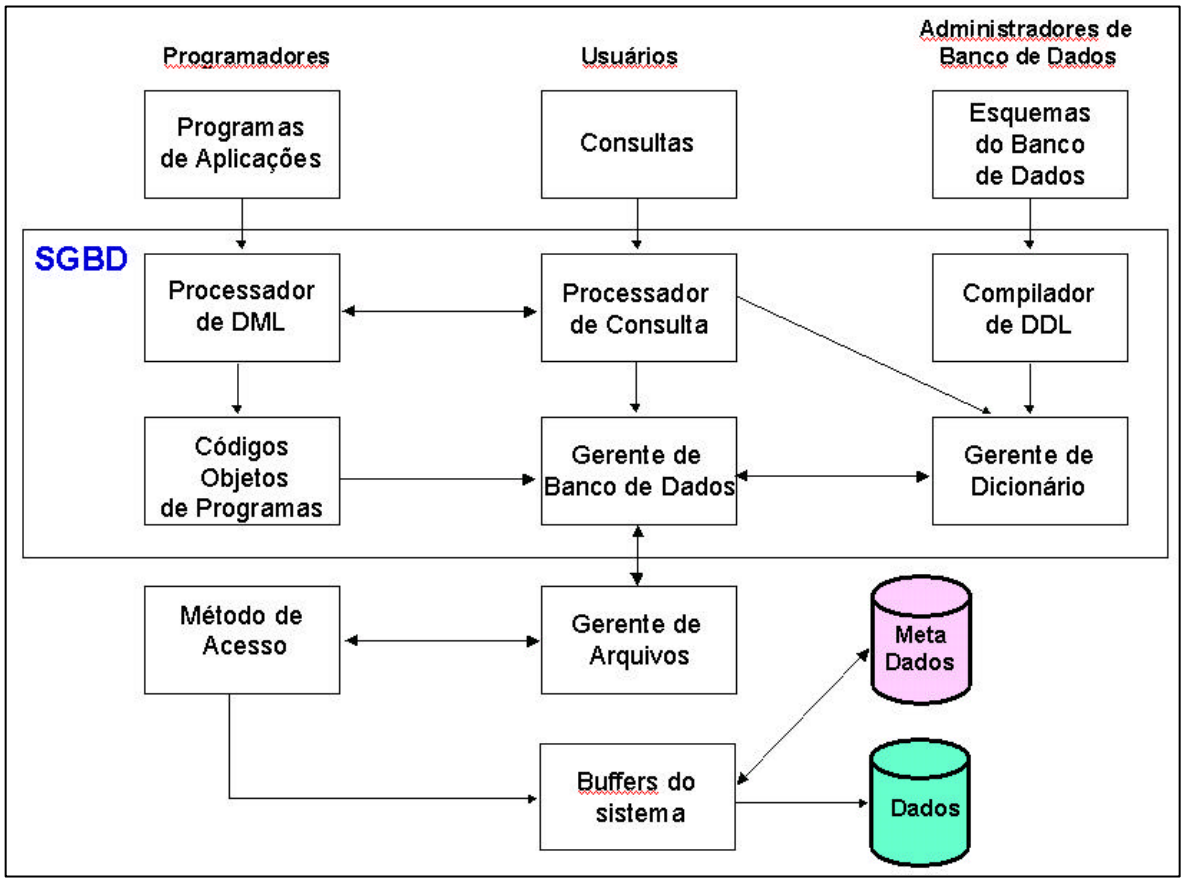


Figura 1: Componentes de um SGBD

⇒ ***Processador de DML***: Esse módulo converte os comandos DML (Data Manipulation Language) embutidos nos programas de aplicação em chamadas de funções padrões das linguagens hospedeiras. Interage com Processador de Consulta para gerar o código apropriado para a consulta.

⇒ ***Compilador de DDL***: Converte os comandos de DDL (Data Definition Language) em um conjunto de tabelas contendo as meta-informações (metadados). Essas tabelas são armazenadas no dicionário de dados, enquanto informações de controle são armazenadas em arquivos *headers*.

⇒ ***Gerente de Dicionário***: Gerencia o acesso e mantém os dados no dicionário. O dicionário é acessado por muitos componentes do SGBD.

Um dos principais componentes de um SGBD é o módulo Gerente de Banco de Dados. Por sua vez, este módulo tem os seus próprios componentes (Figura 2):

⇒ ***Controle de Autorização***: Este módulo verifica se o usuário possui a necessária autorização sobre o recurso solicitado para executar a operação desejada.

⇒ ***Processador de Comando***: Após a verificação e certificação de que o usuário possui a autorização necessária para utilizar o recurso, o controle é passado para este módulo que coordenará a execução da instrução.

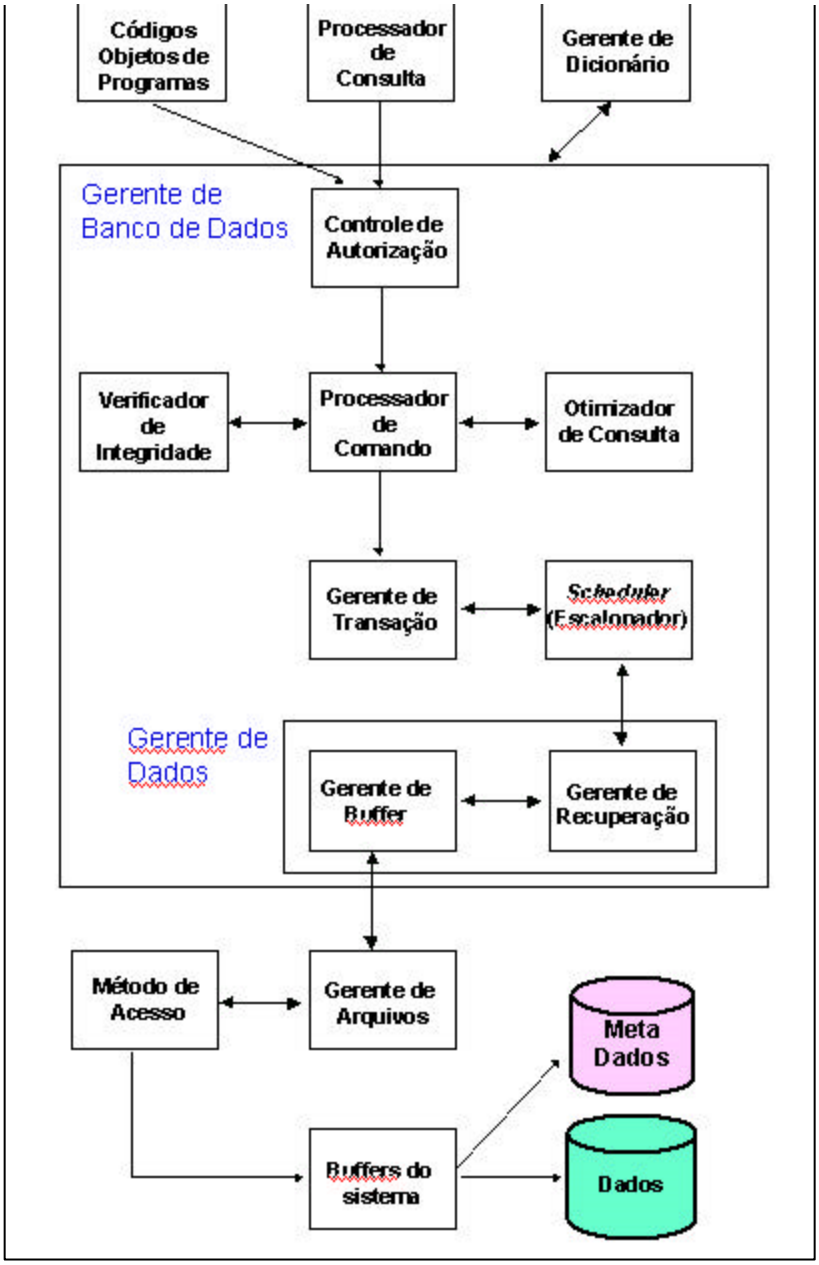


Figura 2: Componentes do módulo Gerente de Banco de Dados

⇒ ***Verificador de Integridade***: Para as operações que alteram o banco de dados, este módulo verifica se a operação desejada irá satisfazer todas as restrições do banco de dados, tais como integridade referencial, chaves primárias e domínios de atributos (*check constraints*), entre outras.

⇒ ***Otimizador de Consulta***: Este módulo determina a estratégia ótima para execução da consulta.

⇒ ***Gerente de Transação***: Este módulo executa os comandos necessários pelas operações recebidos das transações.

⇒ ***Scheduler (Escalonador)***: Este módulo é responsável por assegurar que as operações processadas no banco de dados ocorram sem conflitar com outras operações. Este

controle é relativo à ordem em que as operações das transações são executadas, garantindo a serialização das mesmas.

⇒ ***Gerente de Recuperação***: Este módulo é responsável por assegurar que o banco de dados permanece em um estado consistente em casos de falhas lógicas ou físicas no sistema. É responsável por efetuar o *commit* (confirmação) ou *rollback* (cancelamento) da transação.

⇒ ***Gerente de Buffer***: Este módulo é responsável por transferir os dados entre a memória principal e o armazenamento secundário, tal como disco ou fita. Os módulos Gerente de Recuperação e Gerente de Buffer são freqüentemente referenciados, coletivamente, como Gerente de Dados (*Data Manager*).

Além dos módulos citados, existem muitas estruturas de dados que são necessárias como parte da estrutura física para implementação dos bancos de dados. Essas estruturas incluem arquivos de dados e índices e dicionários de dados.

2.2. SGBD com Regras Ativas

SGBD são ditos passivos, isto é, os dados são criados, recuperados, modificados e excluídos somente em resposta a operações feitas por usuários ou programas aplicativos. Já os SGBDRA executam algumas operações automaticamente, em resposta à ocorrência de certos eventos, como a exclusão de uma linha de uma tabela, ou para satisfazer algumas condições, como garantir o conjunto de restrições de integridade.

SGBDRA possibilitam o desenvolvimento de bancos de dados não convencionais, dando suporte a uma semântica que reflete o comportamento com base em eventos ou acontecimentos num determinado domínio do banco de dados [Pat98].

Os SGBDRA são semanticamente mais expressivos pois podem:

- Realizar funções que em sistemas passivos precisam ser codificadas em aplicações, como manter a réplica atualizada de um dado em outro *site*;
- Facilitar aplicações antes desenvolvidas dentro do escopo de sistemas passivos, como aplicações de regras de negócios;
- Realizar tarefas que requerem subsistemas com “objetivos especiais” em sistemas passivos de banco de dados, como manutenção de integridade de dados entre sistemas.

2.2.1. Regras em Bancos de Dados Ativos

Os SGBDRA são centrados na noção de regras que especificam o comportamento ativo desejado. De forma geral, as regras dos SGBDRA são compostas por três partes:

- **Evento**: faz a regra ser disparada (*triggered*)
- **Condição**: é verificada quando a regra é disparada

- **Ação**: executada quando a regra é disparada e se a condição for verdadeira

Existe uma grande variedade de alternativas para eventos, condições e ações porém, em geral, estas envolvem operações usuais de acesso e manipulação, internas aos bancos de dados. Outros componentes, específicos para tratar regra, também podem ser incluídos, como por exemplo disparos semânticos, utilização do estado de transição anterior ou posterior do banco de dados.

Definido um conjunto de regras, o sistema passa a monitorar os eventos relevantes. Para cada regra, se o evento ocorre, então o sistema avalia a condição da regra e, se for verdadeira, então a ação da regra é executada. Os detalhes de monitoramento e execução de regras são mais complexos, na prática, do que essa simples descrição e variam consideravelmente de sistema para sistema. Entretanto, o paradigma geralmente usado por todos os SGBDRA é este.

Eventos podem ser alterações nos valores dos dados no banco de dados, relevantes para uma aplicação ou um negócio específico. A relevância pode ser mensurada em função da existência de algum tipo de ação que deva ser executada como resultado da ocorrência do evento. Conseqüentemente, um evento pode ser visto como uma situação específica na qual uma ou mais reações podem ser necessárias. Como exemplos, podemos citar a manutenção de integridade referencial e o controle de réplicas e/ou fragmentos em bancos de dados distribuídos.

Esse comportamento, onde são necessárias ações em resposta a um determinado acontecimento, pode ser descrito como o comportamento reativo, envolvendo:

1. A **definição** do acontecimento de interesse (um evento) e uma reação associada ao mesmo;

2. **Detecção** de quando esse evento relevante acontece;

3. A execução da **reação** ao evento.

Sistemas Ativos, de um modo geral, envolvem a detecção do evento e a execução da ação. A resposta ao evento, ou seja, uma ação associada ao mesmo, é executada através de um módulo de execução de regras, de acordo com as reações definidas pela aplicação. SGBDRA são exemplos de Sistemas Ativos que reagem e monitoram circunstâncias específicas relevantes para uma determinada aplicação.

Tradicionalmente, os SGBD são passivos no sentido de que as operações sobre os dados são executadas pelo gerenciador somente quando solicitadas por programas aplicativos ou diretamente pelos usuários. Algumas situações exigem outras operações, reações imediatas que efetivamente não podem ser atendidas por sistemas passivos.

Sistemas de Bancos de Dados utilizando um SGBDRA dão suporte a essas aplicações através do comportamento reativo que essas aplicações exigem do SGBD. Isso implica que o SGBDRA deve prover algum mecanismo para que os usuários possam descrever esse comportamento reativo, além de dar suporte para monitoração e reação em circunstâncias relevantes, assim como para manutenção, consulta e depuração do comportamento reativo. Como conseqüência, os métodos de projeto (concepção) das aplicações devem ser

estendidos para capturar e expressar essa funcionalidade reativa [Pat98]. O atendimento dos requisitos específicos dos SGBDRA implica no desenvolvimento de SGBD estendidos, bem mais complexos que os SGBD convencionais atuais.

2.2.2. Linguagem de Regras

O comportamento desejado de um SGBDRA é especificado usando-se uma linguagem de regras. As definições da sintaxe da regra e conseqüentemente de sua semântica, especificadas nessa linguagem, têm naturalmente um impacto direto sobre o poder de expressividade do SGBDRA.

2.2.2.1. Eventos

O evento especifica o que faz com que a regra seja disparada (*triggered*). Como eventos de disparos típicos, podemos destacar:

- **Modificação de dados** - Um evento de modificação de dados pode ser especificado como uma das três operações de manipulação de dados da SQL: ***insert***, ***delete*** ou ***update*** - em tabela de um banco de dados relacional;

- **Recuperação de dados** - um evento de recuperação (acesso) de dados pode ser especificado como uma operação de seleção de dados da SQL: ***select*** - em tabelas de um banco de dados relacional;

- **Tempo** - Um evento temporal pode especificar que uma regra seja disparada (*triggered*) ora em um determinado momento (por exemplo, 10 de janeiro de 1999, às 12:00 horas), ora repetidamente (por exemplo, todos os dias às 12:00 horas) ou em intervalos periódicos (por exemplo, a cada 10 minutos);

- **Definido por aplicação** - Os eventos definidos por programas aplicativos devem ser especificados, permitindo-se que uma variável seja declarada como representação de um evento (por exemplo, o *login* de um usuário) e que as regras ativas especifiquem tal variável de programa na definição de seu evento de disparo.

Os eventos citados anteriormente são considerados eventos simples, visto que os eventos de disparo também podem ser compostos, seja por combinações de eventos simples e/ou de outros eventos compostos. Para tal utiliza-se:

- **Operadores lógicos usuais** - Os eventos podem ser combinados usando-se operadores *and*, *or*, *not*, *implies*, etc.

- **Seqüência** - Uma regra pode ser disparada quando dois ou mais eventos ocorrerem numa determinada ordem.

- **Composição temporal** – O disparo de uma regra pode ser definido por uma combinação de eventos temporais e atemporais, tais como “cinco segundos depois de um determinado evento”, ou a “cada hora depois da primeira ocorrência de um determinado evento”.

Eventos compostos complexos podem ser definidos se for usada uma linguagem de especificação baseada em expressões regulares ou numa gramática livre de contexto. Finalmente, alguns SGBDRA permitem que o evento seja totalmente omitido, significando que são regras cujas condições devem sempre ser verificadas.

2.2.2.2. Condições

Em uma regra ativa, a condição especifica que um determinado estado do banco de dados seja verificado assim que a regra for disparada e antes da ação ser executada. Como condições mais freqüentes pode-se destacar:

- **Predicados de banco de dados –** Especificados como no caso das cláusulas WHERE da linguagem SQL ou expressão do cálculo relacional.
- **Predicados restritos** – As expressões definindo as condições podem se restringir a comparações de atributos e valores, não especificando junções ou agregados.
- **Consultas de banco de dados -** A condição pode ser representada como uma consulta utilizando-se a linguagem de consulta do SGBD em uso, que é considerada como condição verdadeira se retornar algum resultado.
- **Procedimentos de aplicação -** Uma condição de regra pode ser especificada como uma chamada a um procedimento escrito em uma linguagem de programação da aplicação.

Em todos estes casos, se a linguagem da regra permite eventos parametrizados, a condição inclui provavelmente um mecanismo que possa referenciar valores ligados aos parâmetros do evento. A maioria dos SGBDRA permite que a parte condicional da regra seja omitida, situação na qual a condição é considerada como sendo sempre verdadeira.

2.2.2.3. Ações

Em uma regra de banco de dados ativo, a ação é executada quando a regra é disparada e sua condição é verdadeira. Como ações usuais pode-se destacar:

- **Operações de modificação de dados -** Um SGBDRA relacional pode permitir que ações da regras especifiquem operações de inclusão, exclusão ou modificação de dados.
- **Operações de recuperação de dados** - As ações da regra podem especificar operações de consulta e exibição de dados.
- **Outras operações de banco de dados** - Além da modificação e recuperação de dados, a maior parte dos sistemas suportam operações para definição de objetos (como por exemplo, criação de regras ativas), operações para controle de transações (por exemplo, *rollback* e *commit*), operações para conceder ou revogar privilégios sobre os objetos do banco de dados (*grant*, *revoke*), etc.

- **Procedimentos de Aplicação** - Uma ação de regra pode ser especificada como uma chamada para um procedimento escrito numa linguagem de programação de aplicações, onde o procedimento pode ou não acessar o banco de dados.

Em todos estes casos, se a linguagem da regra permitir eventos parametrizados, então provavelmente inclui um mecanismo para referência de valores ligados aos parâmetros do evento. Se a linguagem permitir que valores sejam passados da condição para a ação, então provavelmente inclui um mecanismo *referenciador* destes valores. Muitas linguagens de regra de banco de dados ativos permitem especificar um conjunto de ações, usualmente com uma ordenação, de forma que as múltiplas ações sejam executadas seqüencialmente.

2.2.2.4. Variações nas regras Evento-Condição-Ação

Quando uma regra ativa não tem um evento, pode significar implicitamente que a regra deva ser disparada sempre que um estágio no qual a condição da regra é verdadeira for alcançado, ou pode significar que a regra deva ser disparada sempre que os dados forem modificados de forma que novos dados satisfaçam à condição da regra.

Há inúmeras diferenças entre regras de Evento-Condição-Ação (caso geral), Evento-Ação, Evento-Condição e Condição-Ação. Por um lado, incluir um evento de disparo como parte da linguagem da regra possibilita especificar ações distintas quando uma determinada condição é satisfeita, dependendo de qual evento tenha ocorrido. Por exemplo, em um banco de dados relacional pode-se desejar reagir à violação de uma restrição de integridade diferentemente, dependendo se a violação ocorreu em função de uma nova tupla ter sido inserida ou porque uma antiga tupla foi excluída. Este tipo de comportamento específico à operação não é possível com regras Condição-Ação e é uma das razões pela qual os bancos de dados ativos têm geralmente adotado regras de Evento-Condição-Ação (ECA). Por outro lado, regras Condição-Ação são geralmente consideradas mais declarativas e podem ser mais fáceis de definir, uma vez que o ônus de determinar quando avaliar a condição recai sobre o sistema ao invés do usuário.

2.2.2.5. Valores de Transição

Para entender o conceito de valores de transição no projeto de uma linguagem de regra é necessário saber que quando uma condição de regra é avaliada ou sua ação é executada, isto ocorre, normalmente, devido a uma mudança no estado do banco de dados - uma transição - que faz com que a regra se torne disparada.

Transições podem consistir de mudanças tão pequenas quanto à modificação de uma simples tupla ou objeto, ou tão grandes quanto uma transição de um banco de dados inteiro, ou seja, a ocorrência de pelo menos uma modificação em uma tupla de cada uma das tabelas de um banco de dados.

2.2.2.6. Eventos Lógicos *versus* Eventos Físicos

Os eventos que disparam regras são, geralmente, operações de banco de dados, notificações oriundas de aplicações ou eventos temporais. Em todos estes casos, o evento de disparo corresponde à execução física de alguma atividade. Às vezes pode ser mais útil

que regras sejam disparadas pelo efeito de uma atividade, ao invés de por operações isoladas efetivamente realizadas.

Por exemplo, considere-se um sistema de banco de dados ativo relacional onde várias tuplas sejam inseridas e as mesmas tuplas atualizadas, em uma única atividade lógica. Se as operações realizadas são *insert* e *update,* as regras que especificam eventos de inserção (física) e as regras que especificam eventos de atualização (física) serão disparadas. No entanto, o resultado líquido (*net effect*)[2] de inserir tuplas e então atualizá-las é equivalente a inserir as tuplas atualizadas. Por conseguinte, se a atividade lógica, ao invés da atividade física, for considerada, somente regras especificando eventos de inserções (lógicas) serão disparadas.

Linguagens de regras de banco de dados ativos podem suportar eventos que sejam físicos, lógicos ou ambos. Em geral, quanto mais próximo o evento de disparo estiver de uma atividade física, mais baixo é o nível de abstração da linguagem da regra. Um nível mais baixo de abstração permite um controle mais refinado sobre como as regras se comportam em função de operações de disparo, enquanto um nível de abstração mais alto permite que as regras sejam programadas de modo mais intuitivo.

2.2.3. Semântica de Execução das Regras

Enquanto a linguagem de regra prescreve o que pode ser especificado em cada regra de um banco de dados ativo, a semântica da execução da regra define como o SGBDRA se comporta uma vez que um conjunto de regras tenha sido definido. Este comportamento inclui a própria semântica do processamento da regra bem como o comportamento do processamento da regra enquanto a mesma interage com consultas e transações no banco de dados.

Como base para discussão, um modelo simplificado de execução de regras é apresentado a seguir. Este modelo pode ser estendido e modificado para obter uma semântica completa de execução de regra.

Quando uma ou mais regras estiverem disparadas, então um algoritmo de processamento de regra será invocado. Na sua forma mais simples, esse algoritmo procura repetidamente por uma regra disparada, avalia sua condição e, se esta for verdadeira, executa a ação da regra. Este algoritmo é mostrado na figura 3 a seguir.

Enquanto houver regras disparadas faça:

1. Encontre uma regra disparada **R**;
2. Avalie a condição de **R**;
3. Se a condição de **R** for verdadeira, então execute a ação de **R**;

Figura 3: Algoritmo simplificado para processamento de regras

[2] O efeito líquido de uma transição consiste num conjunto de inserções, exclusões e alterações de tuplas.

2.2.4. Orientação da execução das regras

Uma execução de uma regra ativa é orientada a instância (*instance-oriented*) se a regra for executada uma vez para cada instância (tupla ou ocorrência) do banco de dados que cause disparo na regra ou que satisfaça à condição da regra. Já uma execução é orientada a conjunto (*set-oriented*) se a regra for executada uma vez para todos os casos do banco de dados onde ocorra o disparo da regra ou que satisfaçam à condição da regra.

A diferença no efeito real entre execução orientada a instância e execução orientada a conjunto é sutil. Por exemplo, considerando uma regra R de um banco de dados ativo relacional que é disparada por inserções de tuplas em uma tabela de *Empregados* e supondo que a ação de R atualiza o valor no atributo *Salário* das tuplas inseridas para serem 10 unidades menores que a média dos valores dos *Salários* em todas as tuplas de *Empregados*:

- Com uma execução de regra orientada a conjunto, uma vez que R executa uma vez para todos os empregados inseridos, todas as tuplas de *Empregados* inseridos serão atribuídas com o mesmo valor para o atributo *Salário*.

- Com execução orientada a instância, os empregados inseridos terão valores distintos para o atributo *Salário*.

2.2.5. Modos de Acoplamento

Os Modos de Acoplamento (*Coupling Modes*) discutem o relacionamento entre o processamento da regra e transações de banco de dados. A abordagem mais simples corresponde a uma condição de uma regra disparada ser avaliada e sua ação executada dentro da mesma transação do evento de disparo, no ponto mais próximo (instrução) do processamento da regra. Entretanto, para algumas aplicações, pode ser útil ou mesmo necessário, adiar a avaliação da condição da regra disparada ou a execução de sua ação até o final da transação. Pode também ser conveniente avaliar a condição da regra disparada ou executar sua ação numa transação em separado (uma nova transação).

Um modo de acoplamento pode especificar a relação transacional entre o evento de disparo da regra e a avaliação de sua condição, enquanto que um outro modo de acoplamento pode especificar a relação transacional entre a avaliação da condição da regra e a execução de sua ação. Os modos de acoplamento possíveis são:

- **Imediato** (*Immediate*) - Acontece imediatamente em seguida à avaliação ou disparo da regra, dentro da mesma transação;

- **Postergado** (*Deferred*) - Acontece no ponto de *commit* da transação corrente;

- **Desassociado** (*Decoupled*) - Acontece em uma transação em separado. Este modo pode ser ainda subdividido entre **desassociados dependentes**, onde a transação separada não é criada até o *commit* da transação original, e **desassociados independentes**, quando a transação separada é criada, indiferentemente do *commit* da transação original.

2.2.6. Arquitetura dos Sistemas Ativos

Embora existam várias arquiteturas para SGBDRA, somente as mais referenciadas na literatura serão citadas a seguir.

- **Arquitetura em Camadas (*Layered*) -** Nessa arquitetura, todos os componentes ativos residem em um módulo construído no topo e independente do SGBD, se este for um SGBD passivo convencional. Utiliza um *run-time*, ou seja, os efeitos da utilização da regra são inseridos no momento da execução da mesma.

- **Arquitetura *Built-In* -** Todos os componentes do banco de dados ativo se tornam parte do próprio SGBD. Isto decorre da modificação de um SGBD passivo existente, adicionando funcionalidades ativas ou construindo um SGBD por inteiro. Também utiliza *run-time.*

- **Arquitetura Compilada -** Nessa arquitetura não é necessário uma atividade "*run-time*". Ao contrário, na hora em que procedimentos de uma aplicação ou operações de banco de dados são compilados, eles são modificados para incluir os efeitos de regras de banco de dados ativos.

2.3. SGBD Distribuídos

Um BDD é uma coleção de dados que pertencem logicamente ao mesmo sistema, mas está fisicamente espalhado (distribuído) pelos *sites* de uma rede de computador. Logo, um BDD é uma coleção de vários bancos de dados distribuídos em vários *sites*, interligados por uma rede de comunicação de dados. Um Sistema Gerenciador de Banco de Dados Distribuídos (SGBDD) é definido como um sistema que permite a administração e acesso eficiente de um BDD, tornando esta distribuição transparente para o usuário.

Considerando que o conjunto de *sites* não é vazio e que cada *site* onde reside um BDD possui capacidade de processamento independente de outro *site*, caso aconteça uma ruptura na comunicação do *site* com a rede de teleprocessamento, o banco de dados continua operando e suprindo as necessidades locais. A figura 4 a seguir exemplifica um esquema de um banco de dados distribuído.

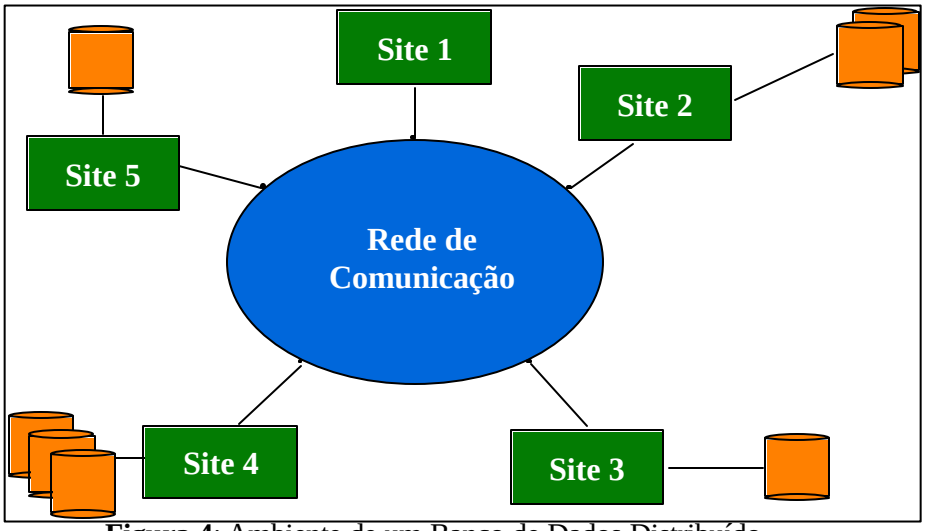


Figura 4: Ambiente de um Banco de Dados Distribuído

Os dados em um BDD são fisicamente distribuídos nos vários *sites* e podem ser fragmentados e replicados. A fragmentação divide o banco de dados em unidades lógicas por diversos *sites* enquanto que a replicação permite que os mesmos dados sejam armazenados em mais de um *site*. Estas técnicas são muito utilizadas durante o projeto do BDD. A informação sobre a fragmentação, alocação dos dados nos *sites* específicos e a replicação são armazenadas em um catálogo do sistema global, fundamental para localização dos dados em um ambiente distribuído.

Existem muitas possibilidades para que um SGBDD implemente as soluções distribuídas. A arquitetura Cliente/Servidor é uma delas, muito utilizada atualmente em sistemas comerciais, onde muitos clientes acessam um único servidor de banco de dados. As arquiteturas em que muitos clientes acessam muitos servidores de banco de dados são mais flexíveis, desde que os bancos de dados estejam distribuídos através dos múltiplos servidores.

Cada máquina cliente possui seu *home* que controla as requisições dos usuários. A comunicação dos servidores de dados entre si é transparente para os usuários executarem suas consultas e transações. Muito freqüentemente são encontrados SGBD implementando uma dessas duas alternativas. A figura 5 a seguir mostra os componentes necessários para implementação de uma arquitetura Cliente/Servidor.

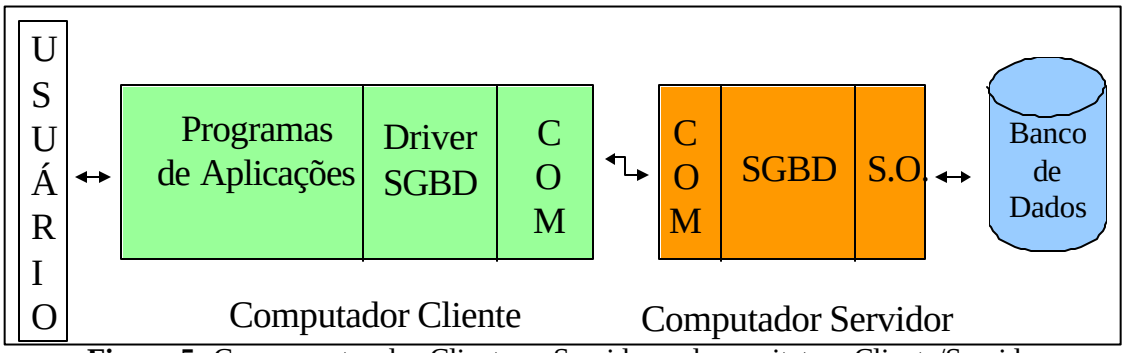


Figura 5: Componentes dos Clientes e Servidores da arquitetura Cliente/Servidor

2.4. SGBD Distribuídos com Regras Ativas

O projeto para processamento de regras em um ambiente de banco de dados distribuído envolve, basicamente, três questões [HSL92]:

- Como decompor uma condição de regra;
- Como distribuir os dados e uma condição de regra entre os diferentes *sites*;
- Como avaliar uma condição de regra num ambiente distribuído e garantir sua consistência e exatidão numa avaliação distribuída.

Com base na distribuição das tabelas do banco de dados, transações do usuário são distribuídas para executarem em múltiplos *sites* ao mesmo tempo. Como premissa básica, o efeito da execução da transação distribuída deve ser equivalente ao efeito da transação se a mesma estivesse sendo executada em um banco de dados centralizado, com as mesmas características do banco de dados distribuído [CW92].

Nestes ambientes, seria desejável que o processamento das regras também executasse de forma distribuída. Um sistema de regra reside em cada *site*, respondendo às mudanças do banco de dados somente naquele *site.* Para obter transparência, o efeito combinado da execução da transação distribuída com o processamento de regras distribuído deve ser equivalente ao efeito combinado da transação original com o processamento de regras centralizado no banco de dados centralizado correspondente. Desta forma, pode-se afirmar que o processamento de regra distribuída é correto

Considerando que o banco de dados distribuído tenha replicação, fragmentação horizontal e vertical [SC98], o SGBD estendido deverá ter um mecanismo especial que garanta a consistência dos dados nas várias cópias distribuídas pelos vários *sites*. Isso implica, em um primeiro momento, que o mesmo deverá ter um grande controle sobre a execução da regra distribuída, além de garantir, também, o controle de todas as transações que atualizarão as réplicas dos dados. Já a fragmentação vertical depende do projeto de regras [SC98].

O SGBD estendido pode usar o paradigma mais abrangente enunciado nessa seção para o processamento de regras ativas num ambiente de banco de dados distribuído e ter seus controles de bloqueios estendidos para serem utilizados em bancos de dados com fragmentação vertical. O paradigma implica na utilização de:

- **Regras Multisite:** Regras que leiam e modifiquem tabelas em *sites* remotos;
- **Início Autônomo:** O processamento de regra seja iniciado independentemente em cada *site* mesmo quando a transação do usuário possa subseqüentemente ler ou modificar tabelas naquele *site*;
- **Prioridades de Regras Intersite:** Prioridades entre regras em *sites* distintos.

3. Framewoks

O reuso de software tem sido um dos principais objetivos da engenharia de software. Reutilizar software não é simples. Com o surgimento do paradigma da orientação a objetos, a tecnologia adequada para reuso de grandes componentes tornou-se disponível e resultou na definição de frameworks orientados a objetos. Frameworks têm atraído a atenção de muitos pesquisadores e engenheiros de software e têm sido definidos para uma grande variedade de domínios. As principais vantagens de framework são, entre outras, o aumento do reuso e a redução do tempo para desenvolvimento de aplicações. Nesta seção são apresentados os principais conceitos relativos a framework, necessários para o entendimento da proposta desse trabalho. Todas as definições apresentadas aqui foram extraídas de [Ucho99].

3.1. Definições

Na literatura da comunidade de orientação a objetos, encontra-se uma grande variedade de definições para framework, com pequenas diferenças entre elas. Três delas são bastante completas e elucidativas:

Bushmann et al. definem framework como sendo um (sub)sistema de software (implementado em uma linguagem de programação), parcialmente completo, criado com o

objetivo de ser instanciado. Um framework define uma arquitetura para uma família de sistemas e fornece blocos básicos predefinidos para sua construção. Também define as partes que devem ser adaptadas para realizar uma funcionalidade específica. Em um ambiente orientado a objetos, um framework é composto de classes abstratas e concretas e sua instanciação consiste de composição e herança de classes. Normalmente, as classes concretas devem ser invisíveis para o usuário do framework.

De acordo com [Ucho99-Pree94] um framework consiste em *frozen spots* e *hot spots.* Frozen spots definem a arquitetura global de um sistema de software – seus componentes básicos e os relacionamentos entre eles. Eles permanecem imutáveis em qualquer instanciação do framework. Hot spots representam aquelas partes do framework que são específicas para cada sistema de software. Hot spots são projetados para serem genéricos – eles podem ser adaptados para as necessidades da aplicação em desenvolvimento. Quando se cria um sistema de software concreto, usando um framework, seus hot spots são preenchidos de acordo com as necessidades e requisitos específicos do sistema.

Já Appleton [Ucho99-Appl97] diz que um framework de software é uma mini-arquitetura reutilizável que fornece a estrutura e comportamento genéricos para uma família de abstrações de software, com um contexto que especifica suas interações e uso dentro de um determinado domínio. Esta especificação é completada através da codificação do contexto, onde as abstrações têm "fim aberto" (*open-ended*), sendo projetadas como *plug-points* específicos. Estes *plug-points* (tipicamente implementados usando *callback* ou polimorfismo) permitem que o framework seja adaptado ou estendido para atender variadas necessidades, e que possa ser combinado com outros frameworks. Um framework não é uma aplicação completa: falta a necessária funcionalidade específica de uma aplicação. Ao invés disso, uma aplicação pode ser construída a partir de um ou mais frameworks, inserindo-se as funcionalidades ausentes nas lacunas *plug-and-play* dos frameworks. Enfim, um framework fornece a infra-estrutura e os mecanismos que executam uma política para interação entre componentes abstratos com implementações abertas.

3.2. Vantagens

Os principais benefícios dos frameworks orientados a objetos decorrem da modularidade, reusabilidade, extensibilidade e inversão de controle que eles oferecem aos desenvolvedores.

Frameworks aumentam modularidade através do encapsulamento de detalhes voláteis de implementação por trás de interfaces estáveis. A modularidade dos frameworks ajuda a aumentar a qualidade do software, concentrando o impacto das mudanças de projeto e implementação, o que reduz o esforço necessário para entender e manter softwares existentes.

As interfaces estáveis fornecidas pelos frameworks aumentam a reusabilidade, definindo componentes genéricos que podem ser reutilizados para criar novas aplicações. A reusabilidade do framework alavanca o conhecimento do domínio e o esforço anterior dos desenvolvedores, a fim de evitar recriação e revalidação de soluções comuns, recorrendo aos requisitos da aplicação e aos desafios do projeto do software. Frameworks maduros permitem uma redução em mais que 90% do volume de código fonte que tem que ser

escrito para desenvolver uma aplicação, quando comparado com software escrito com o suporte de uma biblioteca convencional de funções.

Um framework aumenta a extensibilidade fornecendo métodos "ganchos" (*hook*) explícitos, que permitem que as aplicações estendam suas interfaces estáveis. Métodos hooks desacoplam, sistematicamente, as interfaces estáveis e os comportamentos de um domínio de aplicação, em um contexto particular. A extensibilidade do framework é essencial para assegurar customização adequada de serviços e características para a nova aplicação.

A arquitetura de *run-time* de um framework é caracterizada por uma inversão de controle. Quando se usa um framework, usualmente apenas se implementam umas poucas funções de chamada ou se especializam umas poucas classes, e então se invoca um único método ou procedimento. A partir deste ponto, o framework faz o resto do trabalho, invocando quaisquer chamadas a cliente ou métodos necessários, no tempo e lugar adequados.

3.3. Classificação de Frameworks

Frameworks podem ser classificados em função das técnicas usadas para estendê-los, que variam desde frameworks caixa branca até frameworks caixa preta:

- *Frameworks caixa branca* (frameworks baseados em herança) baseiam-se fortemente em características de linguagem orientada a objetos, como herança e amarração dinâmica, a fim de proporcionar extensibilidade. Uma funcionalidade existente é reutilizada e estendida através de:

  ⇒ Herança de classes base do framework;

  ⇒ Sobrecarga de métodos *hook* predefinidos usando *patterns* como o *Template Method*.

- *Frameworks caixa preta* (frameworks parametrizados) suportam extensibilidade através da definição de interfaces para componentes que podem ser encaixados (*plugged-in*) no framework via composição de objeto. Uma funcionalidade existente é reutilizada através de:

  ⇒ Definição de componentes em conformidade com uma interface particular;

  ⇒ Integração destes componentes no framework usando *patterns* como *Strategy* e *Functor*.

Frameworks caixa branca requerem que o desenvolvedor da aplicação tenha íntimo conhecimento de cada estrutura interna do framework. Embora frameworks caixa branca sejam largamente usados, eles tendem a produzir sistemas que são fortemente acoplados a detalhes específicos das hierarquias de herança do framework. Em contrapartida, frameworks caixa preta são estruturados usando composição de objeto e delegação, ao invés de herança. Como resultado disto, frameworks caixa preta são geralmente mais fáceis de usar e estender do que frameworks caixa branca. Entretanto, frameworks caixa preta são mais difíceis de desenvolver, já que requerem que os desenvolvedores do framework

definam interfaces e *hooks* que antecipem uma grande variedade de casos potenciais de uso.

Freqüentemente, estas duas abordagens de reuso estão simultaneamente presentes em um framework. Características, possivelmente comuns à maioria das aplicações, podem ser oferecidas e portanto reutilizadas como caixas pretas com mínimas mudanças. Por outro lado, a biblioteca de classes que acompanha o framework usualmente fornece classes base (ou caixas brancas) que podem ser especializadas, à medida do necessário, através do acréscimo de subclasses, sendo estas facilmente integradas ao resto do framework.

No início de sua vida, o framework é principalmente concebido como um framework de caixa branca. Entretanto, o framework amadurece à medida que é adotado em uma quantidade crescente de aplicações. Componentes mais concretos, fornecendo soluções do tipo caixa preta para os problemas difíceis, assim como objetos de alto nível, representando as principais abstrações encontradas no domínio do problema, tornam-se disponíveis dentro do framework.

3.3.1. Framelet

Framelet, segundo [Pree99], são frameworks que possuem as seguintes características:

- Não assumem o controle principal da aplicação;
- Possuem no máximo 10 classes/componentes;
- Possuem uma interface simples.

O termo framelet explicita um pequeno framework, normalmente utilizado na construção de outros frameworks. Conseqüentemente, os princípios para construção de framework podem ser aplicados na construção dos framelets.

4. Um Framework de Regras Ativas para um SGBDD

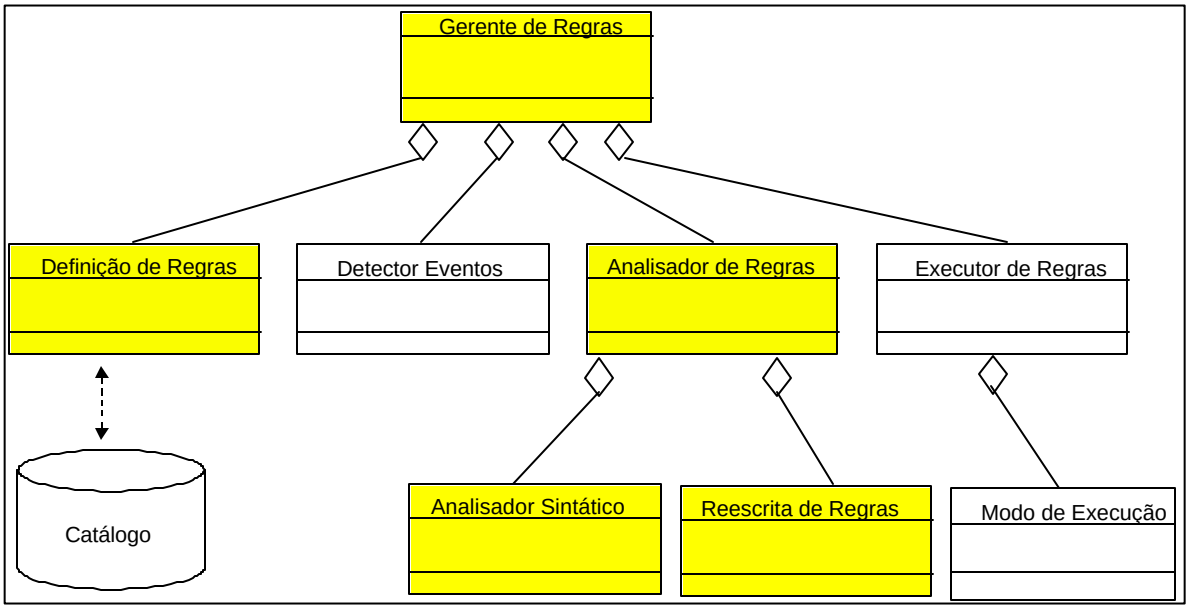
Nesta seção, é proposto um framework para construção de SGBDD com Regras Ativas. São apresentados seus hot e frozen spots e seu uso é exemplificado através de uma instanciação do framework proposto.

4.1. Propósito do Framework

O objetivo do framework é possibilitar a construção de SGBDRA que possam diferir entre si em função da forma como controlam e administram a execução das regras, o modo de detectar seus eventos, bem como quanto ao aspecto de distribuição de regras e dados.

O framework proposto está contido na arquitetura de um SGBD convencional e o estende com características ativas e distribuídas, que podem ser configuradas conforme o interesse. O posicionamento do framework proposto na arquitetura de um SGBD, bem como sua interação com os demais componentes, estão apresentados na Figura 6.

Figura 6: Arquitetura do SGBD com o Framework de Regras Ativas

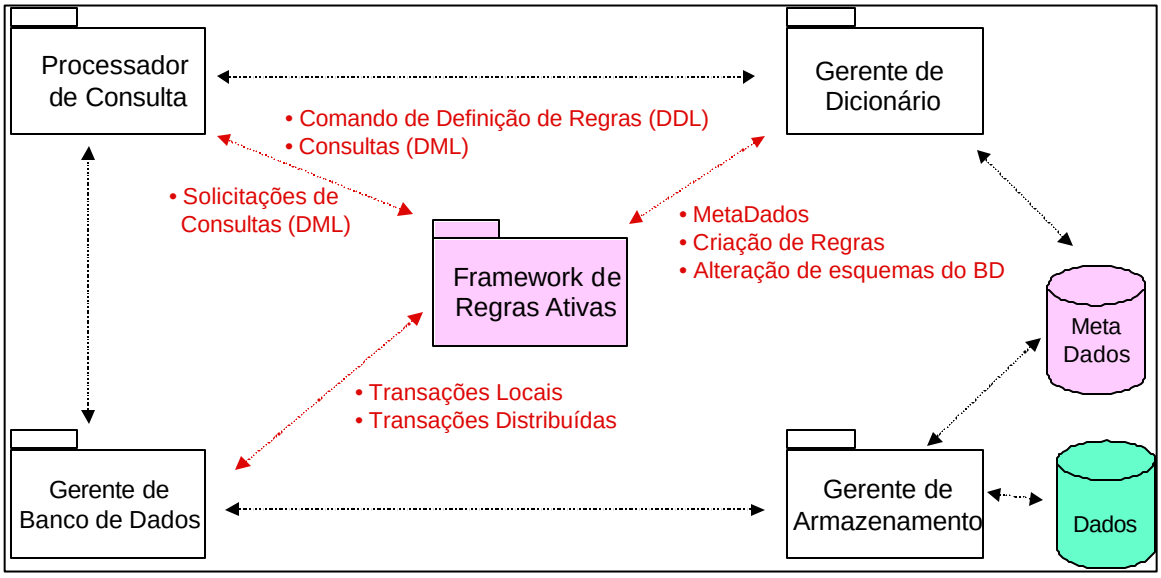


4.2. Componentes do Framework

O Framework de Regras Ativas é composto por um conjunto de componentes. Um de seus componentes é um *framelet*. Ou seja, o framework é composto por um conjunto de componentes e por um outro framework (o framelet)

O Framework de Regras Ativas está apresentado na Figura 7.

Figura 7: Framework de Regras Ativas



Os componentes do Framework de Regras Ativas são**:** *Gerente de Regras*, *Definição de Regras*, *Detector Eventos*, *Analisador de Regras*, *Analisador Sintático*, *Reescrita de Regras*, *Executor de Regras* e *Modo de Execução*. Os componentes [*Gerente de Regras*, *Definição de Regras*, *Analisador de Regras*, *Analisador Sintático* e *Reescrita de Regras*] correspondem aos *frozen spots* do framework, ou seja, conservam-se inalteráveis em qualquer de suas instanciações. Os componentes [*Detector de Eventos*, *Executor de Regras*

e *Modo de Execução*] correspondem aos *hot spots* do framework, ou seja, estas lacunas devem ser preenchidas com as funcionalidades específicas desejadas para uma determinada instanciação do framework. Quanto ao componente *Executor de Regras*, ele pode ser estendido com um componente denominado *Gerente de Distribuição* (maiores detalhes na seção 4.2.2.2). O componente *Gerente de Distribuição*, por sua vez, é um *framelet*: possui seus próprios *hot* e *frozen spots* (maiores detalhes na seção 4.2.2.2.1).

4.2.1. *Frozen Spots* do Framework

Os *frozen spots* (Figura 7) do Framework de Regras Ativas são: **Gerente de Regras**: Componente responsável por todo o controle de análise e execução das regras.

⇒ **Definição de Regras:** Este componente é chamado sempre que uma nova regra é definida. Ele atualiza as informações apropriadas no catálogo do sistema.

⇒ **Analisador de Regras**: Este componente é responsável pelo controle da análise e reescrita de uma regra.

⇒ **Analisador Sintático**: Este componente lê os comandos dos usuários, verifica sua sintaxe e cria uma árvore gramatical, estrutura interna descrevendo o comando do usuário.

⇒ **Reescrita de Regras**: Este componente manuseia as regras implementadas. Combina a árvore gramatical gerada pela consulta do usuário com a regra apropriada e gera uma nova árvore gramatical.

4.2.2. *Hot Spots* do Framework

Os *hot spots* (Figura 7) do Framework de Regras Ativas são:

4.2.2.1. Detector de Eventos

Este componente é responsável por detectar os eventos responsáveis pela ativação das regras. O framework pode ser estendido para detectar diferentes tipos de eventos, como evento de banco de dados (modificação ou recuperação de dados), eventos temporais ou eventos definidos por aplicações:

- **Modificação de dados** - Um evento de modificação de dados pode ser especificado como uma das três operações de manipulação de dados da SQL: ***insert***, ***delete*** ou ***update*** - em tabela de um banco de dados relacional;

- **Recuperação de dados** - Um evento de recuperação (acesso) de dados pode ser especificado como uma operação de seleção de dados da SQL: ***select*** - em tabelas de um banco de dados relacional;

- **Temporal** - Um evento temporal pode especificar que uma regra seja disparada (*triggered*) ora em um determinado momento (por exemplo, 10 de janeiro de 1999, às 12:00 horas), ora repetidamente (por exemplo, todos os dias às 12:00 horas) ou em intervalos periódicos (por exemplo, a cada 10 minutos);

- **Definido por aplicação** - Os eventos definidos por programas aplicativos devem ser especificados permitindo-se que uma variável seja declarada como representação de um evento (por exemplo, o *login* de um usuário) e que as regras ativas especifiquem tal variável de programa na definição de seu evento de disparo.

Para integrar um novo tipo de detector de eventos à arquitetura do Framework, é necessário se definir uma classe especializada da classe *Detector Eventos* da seguinte forma (Figura 8):

Figura 8: *Hot Spot* Detector de Eventos

4.2.2.2. Executor de Regras

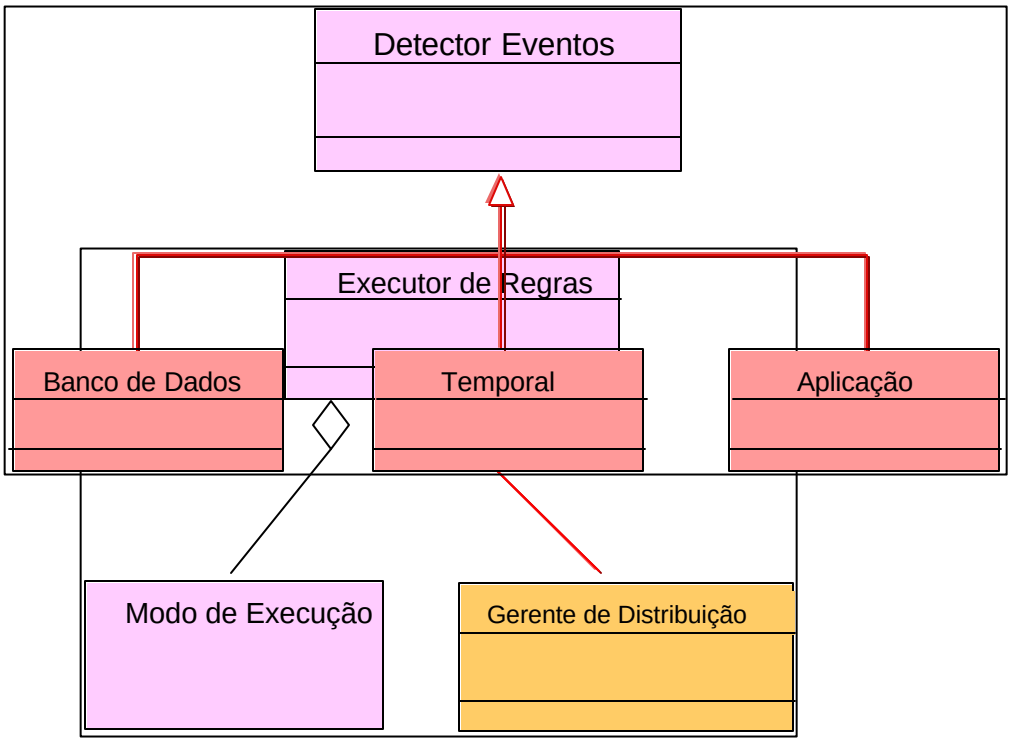


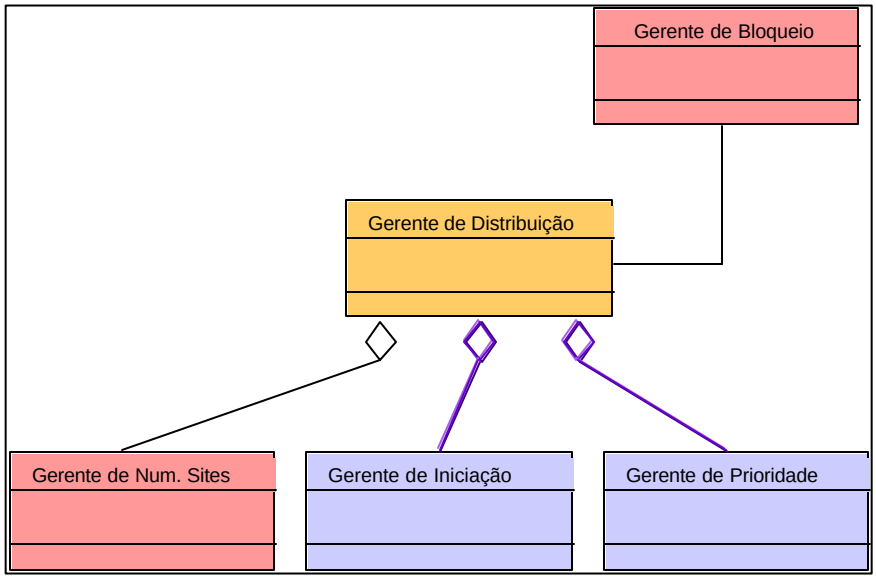
Figura 9: *Hot Spot* Executor de Regras

Este componente é responsável pela execução propriamente dita do plano gerado para regras pelo módulo *Otimizador de Consulta.* Ele é composto pelo componente *Modo de Execução*, responsável pelo controle de análise e execução das regras. O Framework pode ser estendido para executar em um ambiente distribuído, bastando para tal, que seja integrado, ao componente *Executor de Regras*, o componente *Gerente de Distribuição* (Figura 9).

4.2.2.2.1 Framelet Gerente de Distribuição

Este componente é responsável pela execução de regras em ambiente distribuído. Ele também é um framework. Seus componentes são apresentados na Figura 10.

Figura 10: Framelet Gerente de Distribuição



Os componentes do Framelet Gerente de Distribuição são**:** *Gerente de Distribuição*, *Gerente de Num. Sites*, *Gerente de Iniciação*, *Gerente de Prioridade*, *Gerente de Bloqueio.* Os componentes [*Gerente de Distribuição*, *Gerente de Bloqueio*, *Gerente de Num. Sites*] correspondem aos *frozen spots* do framework, ou seja, conservam-se inalteráveis em qualquer de suas instanciações. Os componentes [*Gerente de Iniciação*, *Gerente de Prioridade*] correspondem aos *hot spots* do framework. Eles só deverão ser integrados ao framework caso o processamento das regras possa ser executado em cada *site* independentemente dos demais, o que acarretará na necessidade de controles extras de início das transações das regras e do estabelecimento de prioridades de execução das regras entre os diversos *sites*. Caso tais componentes não sejam integrados ao framework, o processamento das regras só poderá acontecer no mesmo *site* onde ela for de disparada.

4.2.2.3. Modo de Execução

Este componente é responsável por todo o controle de análise e execução das regras. É quem interage com o Gerente de Transação do SGBD para determinar o momento exato da execução das ações relativas às regras. O Framework de Regras Ativas pode ser estendido para especificar diferentes tratamentos da regra quanto ao momento de execução de suas ações, como execução de forma Imediata, Postergada ou Desacoplada do corpo da transação:

- **Imediato** (*Immediate*) - Acontece imediatamente a seguir da avaliação ou disparo da regra, dentro da mesma transação;
- **Postergado** (*Deferred*) - Acontece no ponto de *commit* da transação corrente;
- **Desacoplado** (*Decoupled*) - Acontece em uma transação em separado. Este modo pode ser ainda subdividido entre **desacoplados dependentes**, onde a transação separada não é criada até o *commit* da transação original, e **desacoplados**

independentes, quando a transação separada é criada, indiferentemente do *commit* da transação original.

Para integrar um novo tipo de modo de execução à arquitetura do Framework de Regras Ativas, é necessário se definir uma classe especializada da classe *Modo de Execução* da seguinte forma (Figura 11):

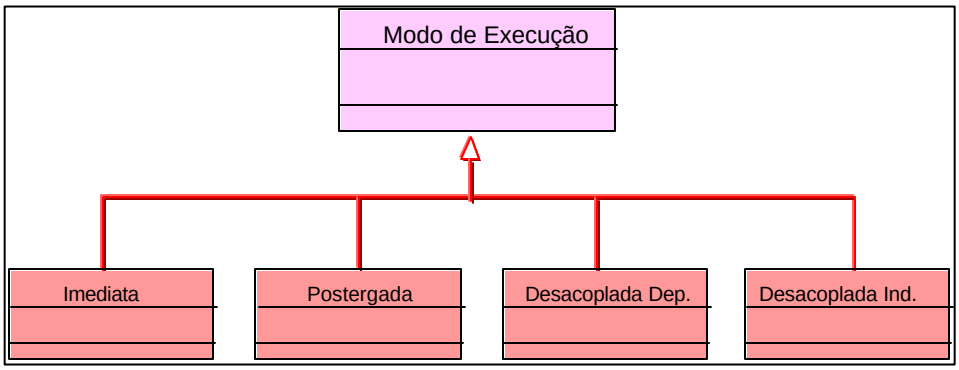


Figura 11: *Hot Spot* Modo de Execução

4.3. Instanciação do Framework

Para exemplificar o uso do Framework de Regras Ativas, será apresentada sua instanciação para a construção de um SGBDDRA específico denominado SGBDDRA-1 (Figura 12-a).

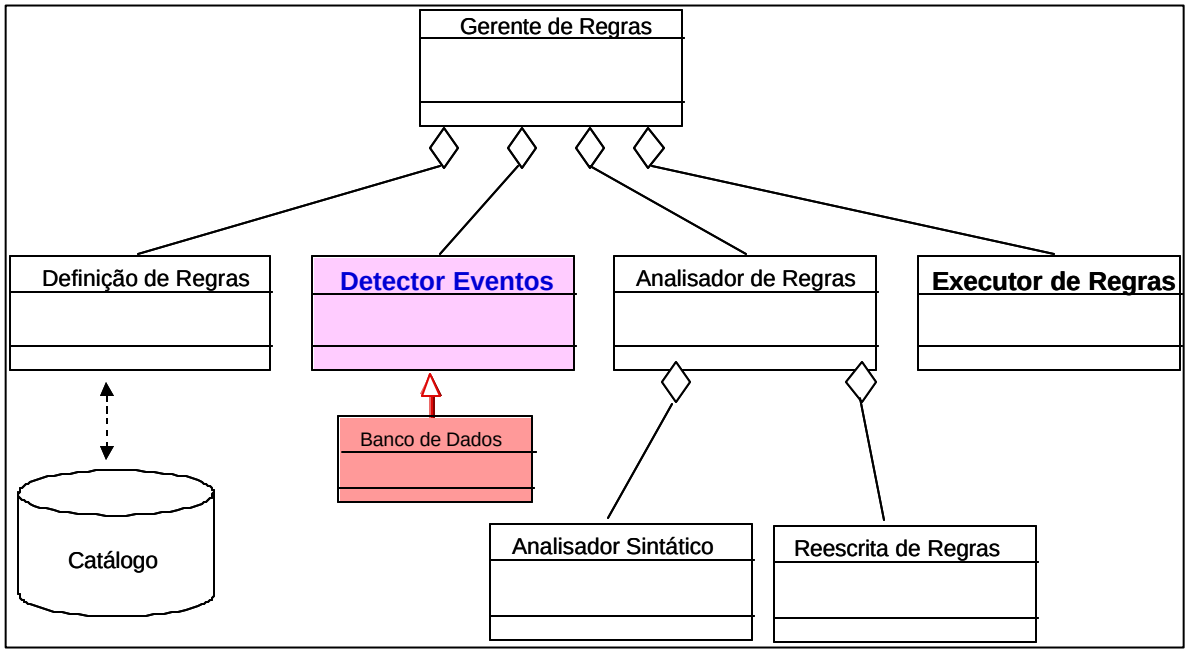


Figura 12-a: Instanciação do Framework de Regras Ativas para o SGBDDRA-1

Como o SGBDDRA-1 funcionará em ambiente distribuído, o framelet Gerente de Distribuição também terá que ser instanciado (Figura 12-b).

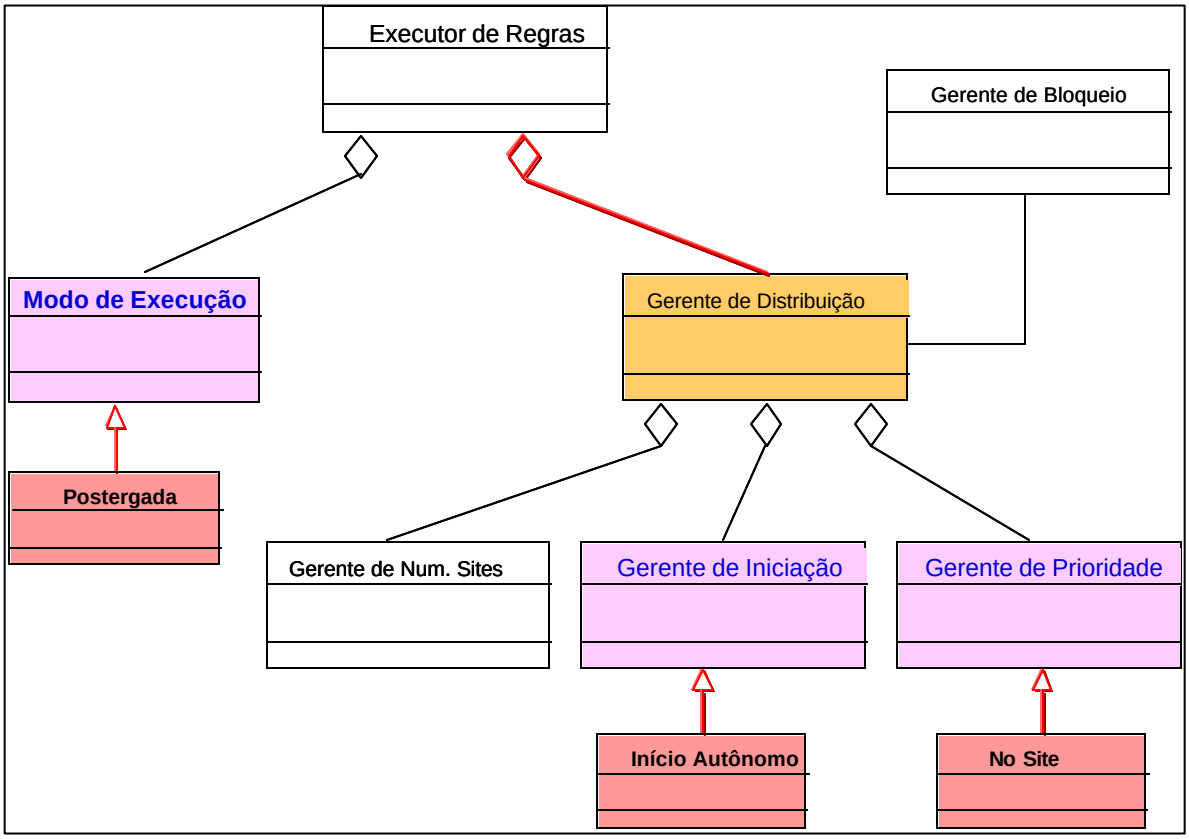


Figura 12-b: Instanciação do *Framelet* Gerente de Distribuição

Na instanciação do Framework apresentada nas Figuras 12-a e 12-b, os *Hot Spots* foram instanciados para satisfazer a um SGBDDRA com as seguintes características:

- **Detecção de Eventos**: Somente eventos de transição de banco de dados serão detectados.
- **Modo de execução das Regras**: As regras serão executadas no ponto de *commit* da transação que realizou o evento.
- **Gerente de Iniciação**: Cada *site* coordenará a iniciação da execução de suas regras, independentemente dos demais *sites*.
- **Gerente de Prioridade:** Cada *site* criará seu próprio *scheduler* (escalonador) para determinar a prioridade de execução das regras.

5. Conclusões e Trabalhos Futuros

A utilização da técnica de framework na construção de SGBDDRA se apresenta como uma alternativa muito eficiente, principalmente em função do reaproveitamento do código das funcionalidades que sejam comuns, enquanto possibilita implementações específicas para diversos ambientes operacionais, com diferentes semânticas de regra.

A proposta desse trabalho foi propor um framework de Regras Ativas que possibilite a construção de SGBDRA que possam diferir entre si em função da forma como controlam e

administram a execução das regras, o modo de detectar seus eventos, bem como quanto ao aspecto de distribuição de regras e dados.

O framework proposto está contido na arquitetura de um SGBD convencional e o estende com características ativas e distribuídas, que podem ser configuradas conforme o interesse.

Diversos outros trabalhos podem dar continuidade a esse estudo, tais como o desenvolvimento de um framework para um SGBDD Paralelo com Regras Ativas, o desenvolvimento dos componentes de terminação e confluência das regras, além da própria implementação do framework proposto.

Referências Bibliográficas

[Cat94] R.G.G. Cattell; “Object Data Management”; Addison-Wesley, 1994

[CW92] Stefano Ceri e Jennifer Widom; “Production Rules in Parallel and Distributed Database Environments”; Proceedings of the 18th International Conference on Very Large Data Bases, Vancouver, Agosto, 1992, pp 339-351.

[CW92a] Stefano Ceri e Jennifer Widom; “Production Rules in Parallel and Distributed Database Environments”; IBM Research Report RJ 8564; IBM Almaden Research Center, Janeiro, 1992.

[CW96] Stefano Ceri e Jennifer Widom; “Active Database Systems Triggers and Rules for Advanced Database Processing”; Morgan Kaufmann, 1996.

[Dat97] C. J. Date; “An Introduction to Database Systems”; Addison-Wesley, 1997, 6th Edition.

[EN94] Ramez Elmasri e Shamkant Navathe; “Fundamentals of Database Systems”; Addison-Wesley, 1994, 2nd Edition.

[EN99] Ramez Elmasri e Shamkant Navathe; “Fundamentals of Database Systems”; Addison-Wesley, 1999, 3rd Edition.

[Lif97] S. Lifschitz; "Sistemas de Banco de Dados Distribuídos e Paralelos"; Anais da I Escola de Informática da SBC - Edição Nordeste (EINE), Novembro, 1997, pp 52-72.

[Mat94] Marta L. Queiros Mattoso; "Sistemas de Bancos de Dados Distribuídos e Paralelos"; Jornadas de Atualização em Informática (JAI), XIV Congresso da Sociedade Brasileira de Computação, 1994, 40 p.

[ÖV91] M.T. Özsu e P. Valduriez; “Principles of Distributed Database Systems”; Prentice Hall, 1991.

[ÖV97] M.T. Özsu e P. Valduriez; “Distributed and Parallel Database Systems”; In: Handbook of Computer Science and Engineering, A. Tucker, 1997.

[ÖV99] M.T. Özsu e P. Valduriez; “Principles of Distributed Database Systems”; Prentice Hall, 1999.

[Pat98] Norman W. Paton; “Active Rules in Database System”; Springer, 1998.

[Pree99] W. Pree; “Framelets as Basis of Lean Component Architectures”; Handouts da palestra proferida na PUC-Rio, 1999.

[Ram97] Raghu Ramakrishnan; “Database Management Systems”; McGraw-Hill, 1997

[SC99] Sérgio da Costa Côrtes; “Regras Ativas em Sistemas de Banco de Dados: Sintaxe, Semântica e Processamento Distribuído”. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 1999.

[SK94] Setrag Khoshafian; “Banco de Dados Orientado a Objeto”; IBPI Press, 1994.

[SKS97] Silberschatz, H. F. Korth e S. Sudarshan; “Database Systems Concepts”, 3d Edition, McGraw-Hill, 1997.

[Ucho99] E.M.A. Uchôa; “Framework para Integração de Sistemas de Bancos de Dados Heterogêneos”. Tese de Doutorado, Departamento de Informática, PUC-Rio, Brasil, 1999.

[VN98] Ioannis Vlahavas e Nick Bassiliades; “Parallel, Object-Oriented and Active Knowledge Base System”; Kluwer Academic Publishers, 1998.

[Wid96] Jennifer Widom; “The Sturburst Active Database Rule System”; IEEE Transactions on Knowledge and Data Engineering, volume 8, número 4, Agosto, 1996.

[ZCF+97] Carlo Zaniolo, Stefano Ceri, Christos Faloutsos, Richard T. Snodgrass, V.S. Subrahmanian e Roberto Zicari; “Advanced Database Systems”; Morgan Kaufmann, 1997.