Integrando Comportamento Ativo a um Sistema de Banco de Dados Orientado a Objetos

Andréa Teles da Silva

Ana Maria de Carvalho Moura

Asterio Kiyoshi Tanaka

Instituto Militar de Engenharia
Depto de Engenharia de Sistemas
Praça General Tibúrcio, 80
22290-270 Rio de Janeiro - Brazil
Telefone: (021) - 2953232 Ramal: 410 - FAX: (021)-5423396
e-mail: teles/anamoura/tanaka/@ime.eb.br

Abstract

Active Database Systems (ADBSs) have been one of the most important database research areas since last decade. Database systems extended to provide active functionalities turn out to be an efficient and uniform mechanism to manage integrity constraints, views and trigger control. ADBSs automatically execute operations in answer to event occurrences and/or required conditions. They are based on the E-C-A (event, condition, action) rule concept, whose structures allow the specification of the desired active behavior. This paper presents the definition of an active rule language for an object manager system - SIGO, conceived from the analysis of the main approaches adopted to implement active systems found in the literature. It describes SIGO active functionalities as well as their main implementation characteristics.

Palavras Chave: Linguagem de Regras, Regras E-C-A, Banco de Dados Ativos

1 Introdução

Uma tendência atual na tecnologia de Banco de Dados (BDs) é estender os sistemas de BDs convencionais de forma a aumentar suas funcionalidades e acomodar cada vez mais aplicações avançadas. Uma mudança bastante útil é transformar os sistemas de BDs em sistemas ativos: o sistema de BD executa por si só certas operações automaticamente, em resposta à ocorrência de certos eventos e/ou condições satisfeitas.

Os Sistemas Gerenciadores de Banco de Dados Ativos (SGBDA) têm como objetivo prover respostas apropriadas a situações pré-especificadas sem comprometer a modularidade do software. O sistema monitora as situações, isto é, os eventos e as condições, e dispara as ações correspondentes quando a condição for avaliada como verdadeira, sem a interferência de usuários ou programas de aplicação.

O paradigma da regra de produção foi generalizado em regras Evento-Condição-Ação (E-C-A) para sistemas de Banco de Dados Ativos (BDAs). Sistemas de BDAs são centrados em torno da noção de regras. Regras nos sistemas de BDAs são estruturas capazes de especificar o comportamento ativo desejado. Na sua forma mais geral, as regras ativas ou regras E-C-A consistem de três partes: Evento: causa o disparo da regra; Condição: é avaliada quando a regra é disparada e Ação: é executada quando a regra é disparada como conseqüência de uma condição verdadeira.

As regras E-C-A ou regras ativas são o suporte para o sistema de BD reagir a mudanças em seu estado e executar ações específicas assincronamente e sem a intervenção de usuários ou programas de aplicações.

O comportamento desejado de Banco de Dados Orientado a Objeto Ativo (BDOOA) é também especificado através de uma linguagem de regras ativas do sistema. Estas regras também expressam comportamento, denominado *comportamento ativo* ou *comportamento reativo* do sistema. Assim, o comportamento em sistemas de BDOOA pode ser expresso de duas maneiras: métodos associados às classes e a ação das regras[BERN94].

SIGO (Sistema Gerenciador de Objetos) é um protótipo em desenvolvimento no Instituto Militar de Engenharia - IME/RJ, que tem por objetivo suportar aplicações emergentes em BDs, provendo um conjunto integrado de ferramentas para ajudar o usuário no desenvolvimento de suas aplicações, através de um ambiente amigável [MOUR95].

Na prática, existem duas abordagens para implementação de um SGBDA: arquitetura em camadas e integrada. No primeiro tipo de arquitetura, uma camada é acrescentada ao topo do sistema para implementar a funcionalidade reativa. Todas as aplicacções que requeiram capacidade ativa têm que interagir com o sistema através desta camada. Esta arquitetura também é referida como acoplamento fraco, uma vez que o processamento e gerenciamento das regras é completamente separado do sistema de Bds.

Os esforços de pesquisa e desenvolvimento em BDAs têm caminhado para a arquitetura integrada. Nesta arquitetura, o SGBD passivo é modificado e estendido para possibilitar o gerenciamento e processamento das regras diretamente integrados ao SGBD.

O objetivo deste trabalho é apresentar a definição e implementação de uma linguagem de regras e o gerenciamento das mesmas no SIGO, tornando-o um Sistema Gerenciador de Objetos Ativo [SILV97], podendo o usuário especificar regras semânticas ou de negócio através de uma interface OO amigável. O sistema utiliza-se do paradigma E-C-A para implementação das regras ativas, fornecendo o suporte necessário para o SIGO monitorar condições especificadas sobre o seu estado e executar ações independentemente de qualquer intervenção do usuário ou programa de aplicação.

Estando as regras sujeitas ao paradigma da OO, estas suportam herança, encapsulamento e composição de regras, além de poderem ser manipuladas como os demais componentes do esquema [MOUR97].

O restante do artigo está organizado da seguinte forma: em 2 descrevem-se as abordagens de implementação de regras ativas em ambientes Orientados a Objetos (OO); em 3 é apresentada a linguagem de regras ativas do SIGO; em 4 são apresentados os principais componentes do SIGO com as suas funcionalidades; em 5 são descritas as características de implementação dos componentes ativos e a visualização do protótipo através de um exemplo executado no próprio sistema; e, finalmente em 6, são feitas as conclusões, com ênfase nas principais contribuições do trabalho.

2 Linguagem de Regras em Ambientes OO

Algumas das diferenças existentes entre as linguagens de regras surgem em função do modelo de dados suportado pelos diferentes sistemas [WIDO96]. É evidente a mudança de paradigma entre o modelo relacional e o modelo OO. Isto justifica o reestudo da funcionalidade, bem como o mecanismo pelo qual a capacidade reativa é incorporada ao modelo de dados OO. Em sistemas relacionais como Ariel [HANS92], Postgres [STON88], Starburst [WIDO91] e produtos comerciais, as regras são definidas e nomeadas como metadados do esquema, juntamente com as tabelas, visões, restrições de integridade e outros. Algumas operações possíveis sobre as regras são: inserir, eliminar e alterar regras. Em sistemas OO como HiPAC [CBB+89], regras são tratadas como objetos de primeira classe e são instâncias do tipo regra definido no esquema. Regras são objetos estruturados, contendo eventos, condições e ações como componentes. Como qualquer outro objeto, as regras podem ser criadas, eliminadas ou modificadas. Além dessas operações, os objetos regra possuem algumas operações especiais tais como: disparar, ativar e desativar.

2.1 Eventos

Várias abordagens são possíveis para especificar eventos no contexto da OO. Atualmente, três abordagens são usadas:

- eventos como expressões declaradas nas definições das classes . como por exemplo, Ode [GEHA91];
- eventos como atributos de regras, O2 [MEDE91]
- eventos como objetos de primeira classe, ADAM [DPG91], SAMOS [GATZ95], Sentinel [ANCH93].

Abaixo, serão examinadas vantagens e desvantagens de cada abordagem.

Eventos como Expressões: Esta abordagem é motivada pelo ganho que se obtém no processamento em tempo de execução, uma vez que o processamento da especificação dos eventos é realizada primeiramente em tempo de compilação. A principal desvantagem desta abordagem é que os eventos não podem ser adicionados, eliminados ou modificados em tempo de execução, resultando em uma dicotomia entre eventos e outros tipos de objetos. Além disso, a persistência de eventos é dependente da existência de outros objetos. Uma outra desvantagem é que os eventos não podem ter atributos e métodos próprios e, conseqüentemente, não podem armazenar e acessar parâmetros computados quando o evento ocorre. Novos tipos de eventos ou atributos de eventos não podem ser incorporados facilmente, e em função disso, a extensibilidade fica comprometida.

Eventos como Atributos de Regras: Considerar eventos como atributos de regra melhora sobre o ponto de vista de permitir que eventos sejam adicionados, eliminados e modificados dinamicamente. Outra vantagem é que a associação de eventos e regras é obtida, uma vez que eventos são parte da estrutura das regras. Entretanto, esta abordagem suporta as mesmas desvantagens da primeira abordagem.

Eventos como Objetos: A última alternativa para especificação de eventos possui muitas vantagens e é superior as outras duas alternativas. Primeiro, esta abordagem modela as propriedades dos eventos. Eventos possuem um estado, estrutura e comportamento, isto é, os eventos possuem as propriedades dos objetos. A informação do estado associada a cada evento inclui a ocorrência do evento e os parâmetros computados quando o evento ocorre. A estrutura de um evento consiste do(s) evento(s) que ele representa enquanto o comportamento consiste em especificar quando o evento é sinalizado. Segundo, eventos podem ser criados, eliminados, modificados e são objetos persistentes como os demais tipos de objetos, isto é, os eventos são tratados de maneira uniforme como os outros objetos. Além disso, novos tipos/atributos podem ser facilmente incorporados pela modificação/acréscimo das definições de classes, sem comprometer a extensibilidade e modularidade do sistema. Entretanto, esta abordagem incorre em sobrecarga em tempo de execução quando eventos são criados, eliminados e modificados dinamicamente.

2.2 Regras

O ambiente OO oferece inúmeras alternativas de projeto para incorporação de regras. As regras podem ser especificadas declarativamente, embutidas dentro de outros objetos como atributos ou dados-membros, ou como objetos. Indiscutivelmente, o mecanismo pelo qual as regras são especificadas em um SGBDOO possui um

profundo impacto na funcionalidade ativa fornecida. A seguir são apresentadas algumas vantagens e desvantagens de cada uma dessas alternativas:

Regras como Declarações somente dentro de Classes: A primeira alternativa de projeto para especificação de regras é uma abordagem declarativa. Regras são declaradas pelos usuários e então inseridas pelo sistema em cada lugar do código que elas podem ser disparadas. É necessário determinar onde e como as regras podem ser declaradas. Regras são associadas a objetos e contribuem para epecificação do seu comportamento. Assim, o lugar propício para se declarar regras é junto a definições de classes. A vantagem principal desta abordagem é que grande parte do processamento da regra é realizado principalmente em tempo de compilação e pouco ou nenhum processamento da regra é executado em tempo de execução. Além disso, a declaração de regras junto a definições de classes oferece um mecanismo para determinar as regras aplicáveis aos objetos. Nesta abordagem, a herança de regras é facilmente suportada.

Esta abordagem não trata as regras como objetos e a existência delas é dependente da existência de outros objetos. Além disso, o sistema não é extensível, uma vez que a adição de novos componentes da regra, por exemplo, níveis de prioridade, requerem modificações nas definições das classes contendo declarações de regras. A principal desvantagem desta abordagem é a ineficiência em manipular a adição, eliminação e modificação de regras. Isto porque alterações nas regras definidas para objetos requerem modificação das definições da classe e recompliação do sistema. Além disso, a modificação de uma definição de classe pode introduzir algumas dificuldades para as instâncias existentes e armazenadas da classe, comprometendo a extensibilidade do sistema uma vez que a adição de regras deve ser permitida aos objetos já existentes do sistema. Nesta abordagem as regras não podem ser reutilizadas ou compartilhadas.

Regras como Atributos: Considerando regras como dados membros, primeiro deve-se achar um tipo de modelo conveniente a elas. Uma vez determinado o modelo apropriado, a principal vantagem desta abordagem é a reusabilidade e extensibilidade. Logo que o tipo tenha sido definido, pode-se usá-lo em diversas aplicações. Além disso, a introdução de um novo componente na regra requer somente a redefinição do tipo definido. As regras são facilmente associadas a objetos já que fazem parte da estrutura do objeto. Nesta abordagem, as regras são adicionadas, eliminadas e modificadas dinamicamente. Entretanto, a principal desvantagem é que não existe suporte a herança. Isto se explica pelo fato do valor de um dado membro não poder ser herdado.

Regras como Objetos: Existem inúmeras vantagens em considerar regras como objetos: i) regras podem ser criadas, modificadas e eliminadas da mesma forma que outros objetos, fornecendo uma visão uniforme das regras no contexto da OO; ii) regras são entidades separadas que existem independentemente de outros objetos do sistema, podendo ser projetadas como objetos transientes ou persistentes. Além disso, estão sujeitas a mesma semântica de transação, assim como os outros objetos; iii) cada regra apresenta um único identificador de objeto, permitindo que as regras sejam associadas a outros objetos; iv) a estrutura e comportamento das regras podem ser ajustados para modelar os requisitos de várias aplicações. Por exemplo, é possível criar subclasses de uma classe de regra e definir atributos especiais ou operações para as subclasses. Finalmente, considerando-se regras como objetos de primeira classe, tem-se um sistema extensível. Portanto, a introdução de novos atributos ou operações na classe regra é trivial, necessitando somente da modificação da definição da classe regra.

3 Especificação da Linguagem de Regras Ativas do SIGO

O Sistema Gerenciador de Objetos (SIGO) [MOUR95] é implementado no topo de um SGBD relacional. Entretanto, considerando-se somente a parte passiva do SIGO, ele provê classes, objetos com identificador, herança, métodos, etc. Ao longo desta seção, a linguagem de regras do SIGO como meio de especificar regras E-C-A é definida. Informações mais detalhadas sobre o sistema SIGO serão abordada mais adiante.

3.3.1 Regras Ativas do SIGO

Regras são construtores básicos de sistema OO ativos. A linguagem de definição de regras do SIGO segue o formato de regra E-C-A conforme ilustrado na Figura 3.1. A definição formal da linguagem bem como a sua BNF encontram-se detalhadas em [SILV97].

Uma vez definido o conjunto de *regras de negócio* , o sistema SIGO monitora os eventos relevantes. Desta forma, ele detecta a ocorrência de cada evento relevante notificando em seguida o componente responsável para a execução desta ocorrência. Conseqüentemente, todas as regras que contém eventos relevantes são disparadas e executadas. Primeiramente, a condição é avaliada, e caso seja verdadeira, a ação é executada.

RASIGO:: = **REGRA** <id_regra>

QUANDO ANTES | DEPOIS <evento_regra>

[**CONDIÇÃO** <condição> < modo_acoplamento>]

AÇÃO <ação> <modo_acoplamento>

PRIORIDADE <prioridade>

Figura 3.1 - Linguagem de Regras do SIGO

A maioria dos sistemas OO ativos trata regras como objetos de primeira classe. Isto significa que as regras podem ser inseridas, eliminadas e modificadas como todos os outros objetos. Dentre as abordagens consideradas na seção 2 para especificar eventos e regras, implementou-se a abordagem que considera eventos e regras como objetos. Todas as *regras de negócio* ou Regras Ativas do SIGO (RASIGO), definidas pelos usuários, são consideradas como classe de objetos. Portanto, elas independem da existência de outros objetos e possuem as vantagens descritas anteriormente. Os eventos das RASIGO são considerados objetos, isto é, são instâncias da classe Evento. As Regras são definidas a nível de classe, isto é, são aplicadas a todas as instâncias da classe e armazenadas como instâncias da classe Regra, possuindo atributos e métodos próprios. Os atributos dos objetos da classe Regra incluem um nome, o modo de acoplamento, prioridades, condição e ação. As instâncias da classe Regra podem ser acessadas pelos métodos ativar e desativar. A Figura 3.2 ilustra parte do esquema das RASIGO.

Sendo SIGO um sistema OO, as regras ativas comportam conceitos a exemplo de classe, identificador de objeto, métodos e herança. As RASIGO especificadas para uma ou mais classes estão sujeitas a herança, e portanto, podem ser propagadas através da hierarquia de classes. Condição e ação podem ser herdadas pelas subclasses e não necessitam ser redefinidas. Uma subclasse pode definir regras adicionais ou redefinir regras herdadas.

3.1 Eventos

Um evento indica um ponto no tempo no qual o sistema deve reagir a uma ocorrência na base de dados, isto é, um fato que ocorre em um dado momento e, teoricamente, não têm duração [TANA95]. As especificações de evento são traduzidas em classe de objetos os quais podem ser criados, eliminados, modificados e persistidos como os outros tipos de objetos, apresentando grande vantagem como já apontado anteriormente.

A Figura 3.3 ilustra a hierarquia da classe Evento. Cada definição de subclasse foi acrescida com os atributos necessários para modelar o tipo de classe que ele representa. Assim, todo evento definido pelo usuário é representado internamente como uma instância da subclasse apropriada.

As RASIGO consistem de 2 componentes: eventos e regras.

3.1.1 Eventos Primitivos

Eventos primitivos podem ser eventos de manipulação de dados e eventos temporais. Eventos primitivos indicam operações primitivas monitoradas pelas regras ativas. As operações de manipulação de dados podem ser: criar, eliminar, recuperar, atualizar objetos e outros métodos definidos pelos usuários.

Na maioria dos sistemas ativos, um evento é concebido como um ponto no tempo quando uma operação de manipulação de dados se inicia ou termina. Em um ambiente OO, entretanto, usuários manipulam objetos através de mensagens enviadas a eles. Assim, cada mensagem origina dois eventos: o ponto no tempo quando a mensagem está "chegando" ao objeto e o ponto no tempo quando o objeto finalizou o método propriamente dito.

A chamada a um método pode ser especificada como um evento (evento_método). Sempre que o método é chamado, o evento correspondente ocorre. Eventos são tipicamente associados a pontos no tempo. Entretanto, a execução de um método requer um intervalo de tempo, e conseqüentemente, a execução de eventos_métodos necessita de uma especificação mais precisa. Isto é feito através do uso de modificadores como ANTES e DEPOIS,

que especificam, respectivamente, o ponto no tempo no qual a chamada a mensagem foi recebida ou o ponto no tempo no qual o método completou a sua execução.

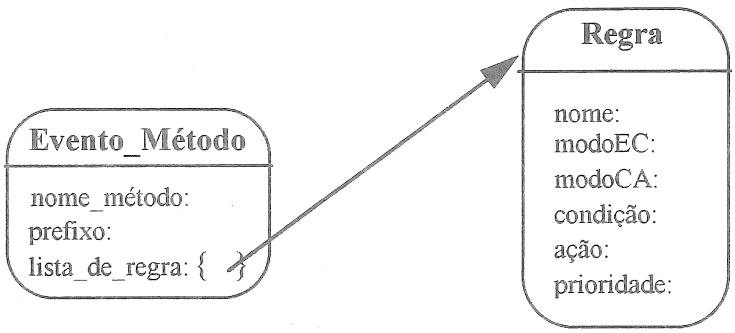

Figura 3.2 - Esquema para eventos do tipo

A chamada a um método pode ser especificada como um evento (evento_método). Sempre que o método é chamado, o evento correspondente ocorre. Eventos são tipicamente associados a pontos no tempo. Entretanto, a execução de um método requer um intervalo de tempo, e conseqüentemente, a execução de eventos_métodos necessita de uma especificação mais precisa. Isto é feito através do uso de modificadores como ANTES e DEPOIS, que especificam, respectivamente, o ponto no tempo no qual a chamada a mensagem foi recebida ou o ponto no tempo no qual o método completou a sua execução.

Eventos temporais são aqueles que podem ser especificados como um ponto explícito no tempo. Os eventos temporais podem ser definidos como absolutos ou periódicos. Um evento temporal absoluto é especificado através de uma data e hora. Por exemplo: 96.12.27,16:15. Um evento temporal periódico utiliza o modificador TODO, por exemplo, TODO DIA,18:00. Neste exemplo, o evento ocorre todo dia às 18:00 horas.

3.1.2 Eventos Compostos

Os tipos de eventos primitivos descritos anteriormente são ocorrências elementares. Entretanto, existem situações que necessitam de eventos complexos. Para atender a esta finalidade, eventos primitivos podem ser compostos em eventos mais complexos, denominados eventos compostos. Eventos compostos são construídos a partir de construtores de eventos. O conjunto de construtores de eventos forma a álgebra do evento. Um evento composto é uma expressão regular construída a partir de eventos primitivos utilizando os construtores de eventos.

Somente a álgebra de eventos simples foi considerada. Consiste de dois eventos construtores: a disjunção e conjunção de dois eventos.

A disjunção de dois eventos, E1 e E2 (denotada por E1 | E2) é um evento composto cuja detecção é sinalizada quando um dos eventos E1 ou E2 ocorreu.

A conjunção de dois eventos E1 e E2 (denotada por E1 & E2) é um evento composto cuja detecção é sinalizada quando os eventos E1 e E2 ocorreram, em qualquer seqüência.

3.2 Regra

O componente regra da RASIGO possui as seguintes propriedades: condição, ação, modos de acoplamento e prioridade. Abaixo, descreve-se cada uma dessas propriedades.

3.2.1 Condição

A condição especifica um predicado a ser avaliado quando a regra é disparada e antes que a ação seja executada.

A cláusula Condição da RASIGO é uma fórmula composta de predicados sobre o estado da base de dados, classes ou objetos do SIGO. Predicados são tipicamente comparações. Os predicados são definidos a partir da Linguagem de Consulta e Manipulação do SIGO (LIMOS) [COST92].

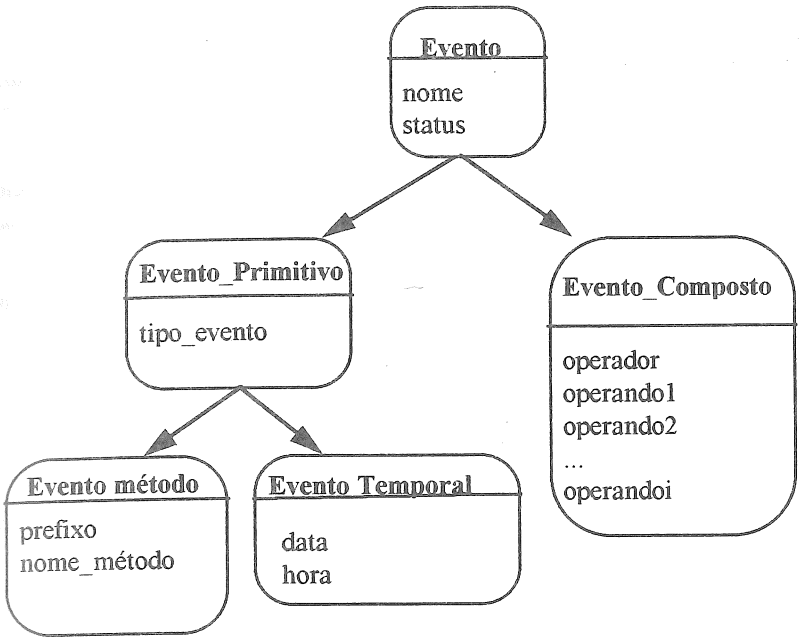

Figura - 3.3 Hierarquia da Classe Evento

Qualquer predicado computável na implementação do modelo de dados pode ser usado como condição das RASIGO. Normalmente, um predicado consiste de predicados primitivos conectados por "AND"ou "OR".

Uma regra sem condição significa que a ação correspondente é executada incondicionalmente toda vez que o evento associado ocorre. A vantagem de se utilizar uma consulta para testar uma condição, ao invés de uma expressão booleana, é que o resultado da consulta pode ser utilizado posteriormente.

3.2.2 Ação

A ação de uma regra é executada quando a regra é disparada e sua condição é verdadeira. Uma ação corresponde a uma seqüência de operações que possam ser codificadas em um método do SIGO. Ações podem ser chamadas a procedimentos externos, bem como uma seqüência de operações de definição e manipulação sobre objetos.

3.2.3 Modos de Acoplamento

O controle sobre a execução de regras disparadas com respeito à transação disparadora acomoda o conceito de modos de acoplamento, que especificam o relacionamento entre o evento disparador e a avaliação da condição ou entre esta e a execução da regra. Cada RASIGO define dois modos de acoplamento: um para o acoplamento evento-condição, e outro para o acoplamento condição-ação. Os modos imediato e retardado foram definidos para os acoplamentos evento-condição e condição-ação.

3.2.4 Prioridade

A cláusula prioridade permite que regras ativas sejam ordenadas conforme definição do usuário, desde que nenhum ciclo seja formado. Durante o processamento, as prioridades especificadas pelos usuários influenciam na escolha da regra a ser selecionada quando mais de uma regra é disparada.

A abordagem a seguir é utilizada quando várias regras compartilharem o mesmo evento: quando um evento é detectado, as respectivas regras são agrupadas em função de seus modos de acoplamento. Entretanto, ainda assim, é necessário estabelecer uma ordem de execução, já que a ação de uma regra pode invalidar a condição de outras. A ordem de execução precisa ser especificada, utilizando-se a cláusula prioridade.

4 SIGO: Uma Visão Geral

Como mencionado anteriormente, esse trabalho foi construído para o sistema SIGO, protótipo de pesquisa concebido como plataforma de experimento para o desenvolvimento de aplicações emergentes em BDs, construído para ambiente Windows, usufruindo das estruturas de acesso e armazenamento de um SGBD relacional.

Para melhor entendimento da parte ativa, faz-se necessário uma revisão suscinta dos principais componentes do sistema, sobretudo aqueles que efetivamente influenciaram no contexto deste artigo: MAME, LIDAS , LIMOS e Gerador de Aplicações.

MAME (Módulo de Aquisição e Modelagem de Esquemas) [GUIM95] é o módulo responsável pelo processo de modelagem de objetos do mundo real, a partir de uma rede semântica para objetos [ROMO91] e de uma interface OO amigável.

Uma vez constituído, o esquema é traduzido em código na linguagem C++, gerando automaticamente métodos genéricos para: construção de classes, recuperação e manipulação dos objetos das classes, e outros. Esta funcionalidade é atribuição do módulo LIDAS (Linguagem para o Desenvolvimento de Aplicações), também responsável em prover ao usuário um ambiente para o desenvolvimento de aplicações [MOUR95].

De modo a oferecer meios de consulta à base de objetos, o sistema oferece uma linguagem de consulta declarativa LIMOS, baseada no padrão SQL, atualmente sofrendo algumas extensões [COLL97].

O sistema inclui ainda um utilitário para geração de aplicações: o módulo Gerador de Aplicações possibilita a geração automática de interfaces e métodos padronizados a partir do esquema definido no MAME, podendo o usuário adaptá-los às suas necessidades [CHAV96].

5 Semântica de Execução

Para suportar a funcionalidade ativa sem comprometer seu desempenho, o sistema SIGO incorpora os seguintes módulos: Detector de Eventos e Gerenciador de Regras, acrescidos à arquitetura do sistema. Estes componentes são responsáveis pela detecção, disparo e gerenciamento das RASIGO.

O módulo Detector de Eventos é responsável pela detecção dos eventos, que é realizada diferentemente, dependendo dos vários tipos de eventos primitivos. No caso do evento primitivo evento_método, o método original é modificado pelo SIGO objetivando a sua detecção. O método original é alterado internamente por um novo, que contém uma chamada à operação Dispara_Regra. A chamada a esta operação pode ser definida antes ou depois do corpo do método original, dependendo das palavras reservadas da linguagem ANTES ou DEPOIS. Para exemplificar tal situação, considere a regra cujo evento é o método Inserir, tendo como modificador a palavra reservada depois: (Quando DEPOIS INSERIR ...). A figura 5.1a ilustra a definição do método Inserir original em uma classe Empregado composta pelos atributos matrícula, função, salário e pesquisa. Na figura 5.1b o mesmo método é automaticamente modificado passando a conter as declarações para a sinalização do respectivo evento quando acionado pela operação Dispara_Regra. Esta função retrata a situação especificada pela Regra Um discutida em 5.1, na qual todo empregado ao ser inserido na base, exercendo a função de analista ou gerente, deverá ter um acréscimo de salário, efetuado pelo método aumenta_sal, que é um método definido na classe empregado no momento em que esta classe é definida no esquema [GUIM95].

O Gerenciador de Regras controla o disparo de regras. Sua interface inclui a operação Dispara_Regra, que é utilizada pelo Detector de Eventos para notificar a ocorrência dos mesmos. O disparo de regras inicia quando o Detector de Eventos sinaliza um evento. O Gerenciador de Regras determina quais regras serão disparadas baseado na informação do modo de acoplamento e prioridade da regra. Pode-se observar que a ordem de execução de regras já é refletida na informação contida no atributo lista_de_regra da classe Evento. Uma vez selecionada a regra, a condição da mesma é avaliada, isto é, faz-se uma consulta à base de dados utilizando-se a linguagem de consulta LIMOS, e caso o seu resultado seja verdadeiro, a ação é executada.

O tipo de arquitetura utlizado para a implementação da funcionalidade ativa no SIGO foi a arquitetura integrada. Nesta categoria, os módulos ativos descritos acima, são integrados diretamente ao SIGO. Uma das vantagens de se utilizar este tipo de arquitetura é a realização eficiente do monitoramento dos eventos, da avaliação da condição e da execução da ação, já que eles ocorrem diretamente dentro do sistema.

```
bool EMPREGADO::INSERIR()
{
        hDBIDb hCur = NULL;
        pBYTE pBuf = NULL;
        int nref = 0;
        if ((PESSOA::INSERIR()))
        {
         if (AbrirTabela("EMPREGADO",&hCur))
         {
          if (PreparaBuffer(hCur,&pBuf))
          {
           DbiPutField(hCur,1,pBuf,(pBYTE)&Surrogate);
           DbiPutField(hCur,2,pBuf,(pBYTE)&nref);
           DbiPutField(hCur,3,pBuf,(pBYTE)&MATRICULA);
           DbiPutField(hCur,4,pBuf,(pBYTE)FUNCAO);
          DbiPutField(hCur,5,pBuf,(pBYTE)&SALARIO);
                                double umPESQUISA = PESQUISA ? PESQUISA->ObterId() : 0;
                                DbiPutField(hCur,6,pBuf,(pBYTE)&umPESQUISA);
                                if (DbiInsertRecord(hCur,dbiNOLOCK,pBuf) == DBIERR_NONE)
                                {

        SomaNref("EMPREGADO","PESQUISA","PROJETO",Surrogate,1);
                                        CloseCursor(&hCur);
                                        ReleaseResources(pBuf);
                                        return true;
                                }
                        }
                }
        }
        CloseCursor(&hCur);
        ReleaseResources(pBuf);
        return false;
}
```

Figura 5.1a -Definição do método Inserir na classe Empregado

```
bool EMPREGADO::INSERIR()
{
                ...
                                        Dispara_Regra("((EMPREGADO.FUNCAO = \"Analista\") OR
(EMPREGADO.FUNCAO = \"Gerente\"))","EMPREGADO",&EMPREGADO ::AUMENTASAL,this);
                                        SomaNref("EMPREGADO","PESQUISA","PROJETO",Surrogate,1);
                ...
}
```

Figura 5.1b -Redefinição do método Inserir na classe Empregado

5 1 Ambiente do Protótipo

De acordo com as especificações apresentadas nas seções anteriores foi implementada a primeira versão do SIGO Ativo. As definições das RASIGO sobre uma base de objetos são apresentadas através de um exemplo implementado sobre o protótipo da aplicação SAD - Sistema Administrativo definido no ambiente do SIGO, no qual será definida a seguinte RASIGO:

- **Regra Um**

Contexto: Ao enviar uma mensagem ao método *Inserir* da classe **Empregado**, verificar se o atributo ou variável de instância *função* desta classe tem valor igual a *"Analista"* ou *"Gerente"* . Caso esta condição seja satisfeita o método *Aumentasal* (aumenta salário) da classe **Empregado** é executado.

As RASIGO são definidas na classe, isto é, a partir da interface do MAME. Ao selecionar uma classe, é possível definir as RASIGO a partir do botão Regras. A figura 5.4 ilustra esta situação. Ao selecionar o botão Regras, o usuário pode, a partir da janela de Interface para Definição de Regras, definir as RASIGO para a classe selecionada. O usuário nomeia a RASIGO e em seguida seleciona o botão Criar Regras conforme ilustra a figura 5.5. A figura 5.6 exibe a interface que permite ao usuário selecionar o método que causará o disparo da regra Um depois da execução do método Inserir. Na figura 5.7, o usuário define a condição a ser avaliada quando o evento_método Inserir for detectado. A figura 5.8 ilustra a definição da ação a ser executada quando o evento Inserir for detectado e a condição especificada for avaliada como verdadeira. Finalmente, a figura 5.9 ilustra a regra Um, segundo o formato ECA.

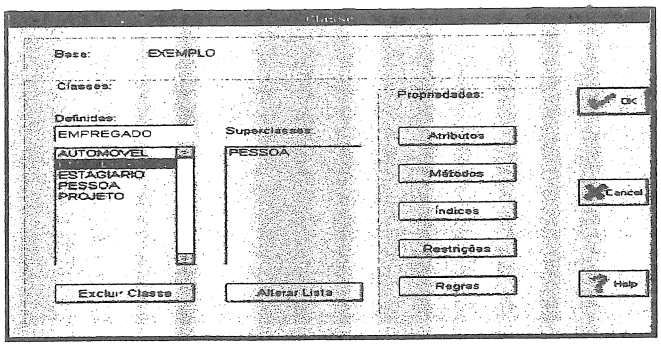

Figura 5.4 - Interface do MAME

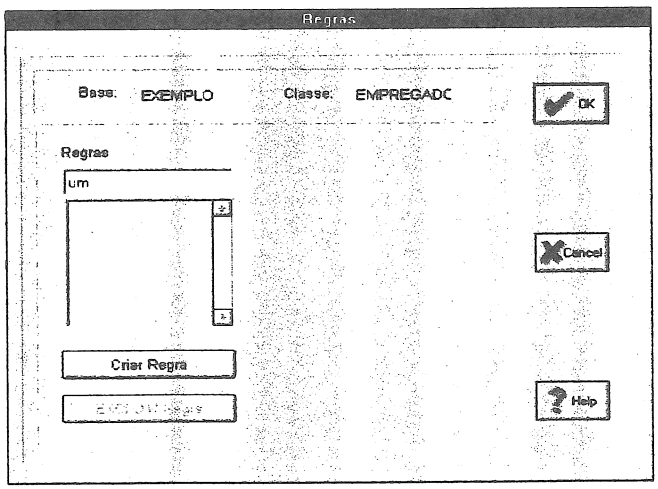

Figura 5.5- Interface de Definição de Regras

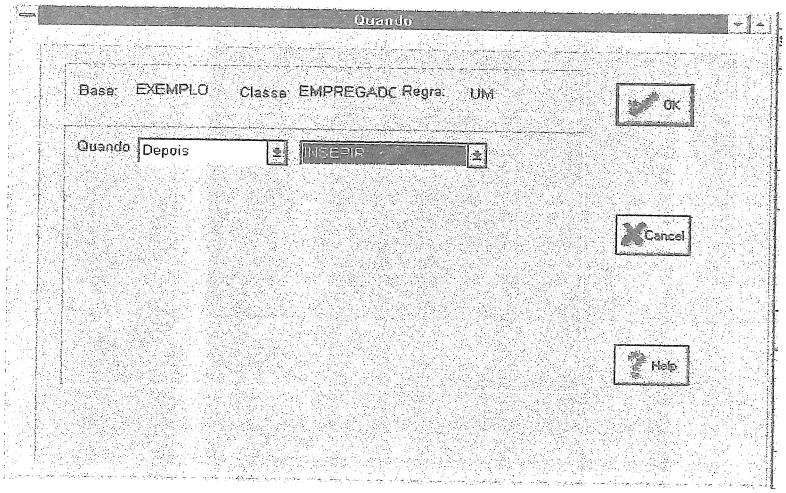

Figura 5.6 - Interface para Definição de Evento

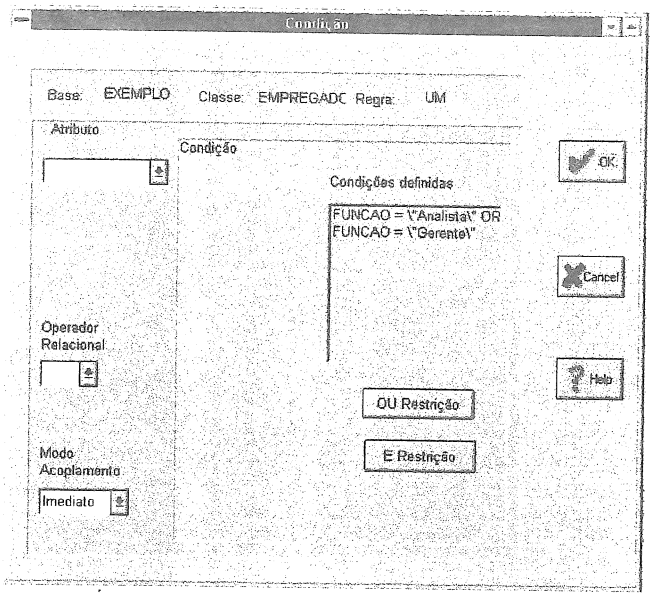

Figura 5.7 - Interface para Definição da Condição

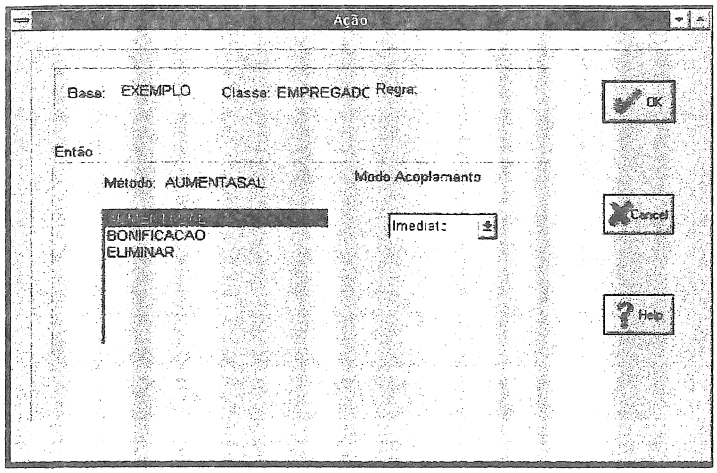

Figura 5.8 - Interface para Definição da Ação

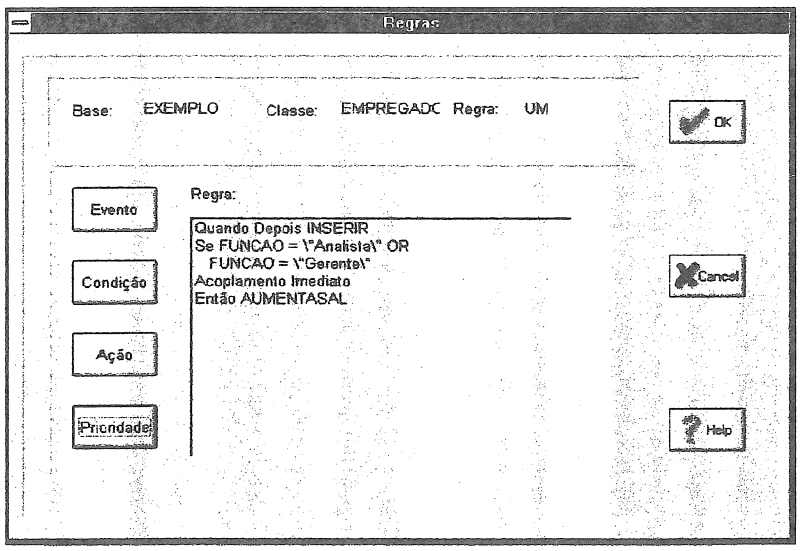

Figura 5.9 - Definição da Regra Um

6 Conclusão

Este trabalho apresentou o resultado de implementação de regras ativas no SIGO, um ambiente de BDOO atualmente em desenvolvimento no IME/RJ. O principal objetivo foi definir e implementar uma linguagem de regras ativas, bem como gerenciar o uso das mesmas no SIGO, tornando-o um sistema ativo.

O comportamento no SIGO pode ser expresso de duas maneiras diferentes e não-ortogonais: *comportamento passivo*, que é representado pelos métodos associados às classes e *comportamento ativo* que é o seu uso sob ação de regras. O tratamento das regras ativas do sistema (RASIGO) como objetos permitiu usufruir de todas as características pertinentes da OO. A adoção da arquitetura integrada na implementação dos componentes ativos, isto é, o detector de eventos e o gerenciador de regras foi vantajoso, uma vez que o monitoramento dos eventos, a avaliação da condição e execução da ação são realizadas eficientemente, já que ocorrem diretamente no próprio sistema.

O próximo passo vislumbrado no sistema é a construção de um conjunto de ferramentas para dar suporte ao usuário durante o desenvolvimento e manutenção das aplicações no SIGO. O "tracer" para regras é uma ferramenta importante, pois permite monitorar o comportamento das regras em tempo de execução. Um outro aspecto a ser considerado em trabalhos futuros é a resolução de conflitos quando várias regras são disparadas, isto é, será necessário determinar uma ordem serial para que múltiplas transações disparadas sejam executadas.

Referências

[ANCH93] Anwar E., Maugis L., Chakravarthy S.; A New Perspective on Rule Support for Object-Oriented Databases. In Proceedings of the ACM SIGMOD Int. Conference on Management of Data, 1993.

[BERN94] Berndtsson M., Lings B.; Systematic Treatment of Events and Rules. Lou Foster, valparaiso University, USA, 1994.

[CBB+89] Chakravarthy S., Blaustein B., Buchmann A, Carey M., Dayal U., Goldhirsch, Hsu M., Jauhari R., Livny M., McCarthy D., McKee R., Rosenthal A .; HiPAC: a Research Project in Active, Time-constrained Database Management. Final report. Technical Report XAIT-89-02, Xerox Advanced Information Technology, 1989.

[CHAV96] Chaves E, Dias A; Gerador de Aplicações para o SIGO: Um Sistema Gerenciador de Objetos. Projeto Final de Curso, IME, nov 1996.

[COLL97] Collyer C., Linguagem de Consulta ao SIGO: um Sistema Gerenciador de Objetos. Tese de Mestrado, IME, Rio de Janeiro, (em andamento).

[COST92] Costa, M. R.; Uma Linguagem de Consulta para um Banco de Dados Orientado a Objeto. Tese de Mestrado, IME, Rio de Janeiro, jul 1992.

[DPG91] O. Diaz, N. Patom, and P. Gray.; Rule Mangement in Object-Oriented Databases: a Uniform Approach. In Proceedings of the Intl. Conference on Very Large Databases, 1991.

[GATZ95] Geppert A., Gatziu S., Dittrich K. R., Fritschi, Vaduva A. Architecture and Implementation of the Active Object-Oriented Database Management System SAMOS- htttp://www.ifi.unizh.ch/groups/dbtg/dbtg-papers-html.

[GEHA91] Gehani N. H., Jagadish H. V.; Ode as an Active Database: Constraints and Triggers. Proceedings of the 17th Int. Conference on Very Large DataBases, Barcelona, set 1991.

[GUIM95] - Guimarães A. R. N.; Interface Orienteda a Objetos para Criação e Manipulação de Esquemas Conceituais no SIGO. Tese de mestrado, IME, Rio de Janeiro, março 1995.

[HANS92] Hanson E. N.; Rule Condition Testing and Action Execution in Ariel. In Proceedings of the ACM SIGMOD International Conference on Management of data, 1992.

[MEDE91] Medeiros C. B., Pfeffer P. Transformação de um Banco de Dados Orientado a Objetos em um Banco de Dados Ativo. Em Anais do Simpósio Brasileiro de Banco de Dados, Manaus, AM, mai 1991.

[MOUR95] Moura A. M. C. et al; Dealing with Persistence in an Object Management System Using Relational Implementation. XV Int. Conference of the Chilean Computer Science Society, Arico, Chile, nov 1996.

[ROMO91] Rossetti L. C., A. M. C.; Modelagem de Esquemas em um Ambiente de Banco de Dados OO. XVIII Conf. Latino americana de Informática, (CLEI91), Venezuela, Caracas, jul 1991.

[SILV97] Silva T. A. Implementação de Regras Ativas em um Sistema Gerenciador de Objetos. Tese de Mestrado, IME, Rio de Janeiro.

[STON88] Stonebraker M., Hanson E. N., Potamianos S.; The POSTGRES Rule Manager. IEEE Transactions on Software Engineering, 14(7), 1988.

[WIDO91] Widom J., Cochrane R. J. and Linsday B. G.; Implementing Set-Oriented Production Rules as an Extension to Starburst. In Proceedings of the 17th Intl. Conference on Very Large DataBases, pages 275-285, Barcelona, Spain, set 1991.

[WIDO96] Widom J., Ceri S.; Active Database Systems - Triggers and Rules for Advanced Database Processing. Morgan Kaufmann Publishers, San Francisco California, 1996.