SIDNEY DA SILVA VIANA

UMA LINGUAGEM DE GERÊNCIA DE REGRAS COMO EXTENSÃO DA LINGUAGEM SQL3

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Doutor em Engenharia Elétrica.

São Paulo

2007

SIDNEY DA SILVA VIANA

UMA LINGUAGEM DE GERÊNCIA DE REGRAS COMO EXTENSÃO DA LINGUAGEM SQL3

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Doutor em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Jorge Rady de Almeida Junior

São Paulo

2007

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com anuência de seu orientador.

São Paulo, 15 de Agosto de 2007

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Viana, Sidney da Silva
Uma linguagem de gerência de regras como extensão da linguagem SQL3 / S. da S. Viana. -- São Paulo, 2007.
174 p.

Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Banco de dados ativos 2.Gerência de regras 3.SQL3 I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

DEDICATÓRIA

Aos meus pais

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Jorge Rady Almeida Junior, pela orientação dedicada e pelos valiosos comentários, críticas e sugestões.

A prof. Dra. Edit Grassiani Lino de Campos, que me ensinou o caminho a ser trilhado.

A minha família, pelo apoio e incentivo constantes, motivações essenciais para que eu chegasse à fase final.

Aos colegas pelo incentivo e colaboração.

A todos que colaboraram direta ou indiretamente para a conclusão deste trabalho.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A - Ação

BD - Banco de Dados

CA – Condição-Ação

CA_1A_2 – Condição-AçãoPrimária-açãoSecundária

ECA - Evento-Condição-Ação

ECA_1A_2 – Evento-Condição-AçãoPrimária-açãoSecundária

LDD – Linguagem de Definição de DAdos

LMD – Linguagem de Manipulação de Dados

SAMOS - Swiss Active Mechanism-Based Object-Oriented Database System

SBD – Sistema de Banco de Dados

SBDA – Sistema de Banco de Dados Ativo

SGBD - Sistema Gerenciador de Banco de Dados

SGBDA – Sistema Gerenciador de Banco de Dados Ativo

SQL3 ou SQL99- Structure Query Language

RESUMO

Este trabalho adota um modelo de regras estendido, que melhora a expressividade da linguagem SQL3, propondo o uso de novas variantes para o modelo de regras ECA (Evento – Condição – Ação). Porém, este modelo estendido abrange somente a definição de regras, faltando as outras operações de gerência, como eliminar ou modificar uma regra, entre outros mecanismos necessários para gerenciar estes novos tipos de regras. Neste trabalho, propõe-se uma linguagem de gerência de regras composta de um conjunto de operações para criar, excluir e alterar as regras e suas partes, com a finalidade de obter maior reuso e manutenibilidade das regras. Para tanto, analisa-se o modelo de regras estendido, para identificar quais são as suas limitações e as propriedades do modelo a serem consideradas na especificação da linguagem de gerência de regras proposta. O resultado desta análise é utilizado para a elaboração de um repositório de regras, necessário para armazenar os tipos de regras propostos. Este repositório armazena um conjunto de regras que se deve manter consistente, da mesma forma que os dados se mantêm consistentes em um banco de dados. Para tanto, foi definido um conjunto de regras de consistência. Também, é definido um conjunto de operações de gerência de regras que auxiliam na manipulação de regras e de seus elementos, armazenadas no repositório de regras.

Palavras-chave: Banco de Dados Ativos. Gerência de Regras. SQL3

ABSTRACT

This work uses an extended rule model which improves the expressiveness of SQL3 language, proposing the use of new variants for the ECA (Event-Condition-Action) rule model. Nevertheless, the extended model considers only the rule definition and it lacks other management operations, such as excluding or modifying rules, among others necessary mechanisms for these new rule types. In this work, a rule management language made of a set of operations for creating, eliminating and altering rules and its parts is proposed, in order to obtain greater reuse and maintainability of rules. For this purpose, the extended rule model is analyzed to identify its limitations and the properties that it supports to be considered in the rule management language proposed. The result of this analysis is used to define a rule repository, necessary for storing the rule types proposed. This repository stores a rules set which must be consistent, as well as data must be consistent in the database. Hence, a consistency rule set is defined. Besides, a rule management operations set is defined to help to handle rules and their elements, stored in the rule repository.

Keywords: Active Database. Rule Management. SQL3

CAPÍTULO 1. INTRODUÇÃO

1.1 MOTIVAÇÃO

A relevância da área de Banco de Dados tem sido comprovada, por meio do grande volume de aplicações que utiliza Sistemas Gerenciadores de Banco de Dados (SGBD), como um dos componentes principais, no desenvolvimento de sistemas computacionais. Este sucesso se deve ao fato de que estes SGBD consolidam tarefas de armazenamento e recuperação de grandes volumes de dados eficientemente, aliados à segurança e a ambientes centralizadores. Este sucesso também pode ser constatado, pela própria indústria de software (por exemplo, Oracle, Informix, DB2), que propicia ambientes complexos, que garantem, de forma confiável e eficiente, mecanismos que manipulam, eficazmente, esta grande quantidade de informações.

A evolução dos SGBDs, levou à adoção de capacidades ativas, adotando a semântica destas aplicações (PATON, 1998). Esta semântica, representada por meio de regras, esta refletida em comportamentos baseados em eventos. O evento é algum acontecimento, relevante, no tempo. A relevância pode ser medida pelo fato de que alguma ação deve ser tomada, em função da ocorrência deste acontecimento eventos são elementos essenciais na elaboração de regras. Regras representam sentenças ou restrições associadas aos estados ou processos de um domínio, por exemplo, compra e venda de produtos via *Web*.

O comitê de padronização da linguagem SQL3, incluiu, em sua última versão (ISO/IEC 9075-2, 1999), (linguagem SQL3 ou linguagem SQL99), a proposta de inclusão de **regras** ou *triggers* nos SGBDs, e que por sua vez foi seguida pela indústria de software. A linguagem SQL3 é um padrão de fato, utilizado pelos Sistemas Gerenciadores de Banco de Dados Ativos (SGBDAs) para a definição e manipulação de dados e regras em banco de dados. Um SGBDA é um SGBD, que além de suas características próprias, permite a definição, gerência e execução de regras **Evento-Condição-Ação** (ECA). As regras ECA caracterizam-se por terem três componentes: o **evento**, a **condição**, associada a um predicado a ser avaliado, e a **ação**, que é um conjunto de operações a serem executadas, toda vez que o predicado seja avaliado como verdadeiro.

A Inclusão de regras dentro de um SGBDA leva a uma série de vantagens, como por exemplo:

- **A reutilização de códigos**: regras computacionais derivadas de diferentes domínios de aplicação podem ser especificadas por meio de uma linguagem de programação provida pelo próprio SGBDA. Estes fragmentos de códigos representam a semântica envolvida nos processos de negócios e podem ser armazenados e utilizados por diversas aplicações. Por exemplo, em áreas como *Workflows* no qual a automação de processos de negócio, por meio de tarefas e/ou documentos são passadas de um participante a outro, de acordo com regras procedimentais, *eBusiness*, no qual o foco é a obtenção de maior vantagem competitiva, por meio da cadeia de valor da empresa, tais como seus fornecedores, empregados, acionistas, consumidores e seus processos como conhecimento (obtenção de conhecimento de produtos por parte do cliente, treinamento de revendedores para vender com maior eficácia), transação (compra eletrônica de produtos) e relacionamento (intercâmbio eletrônico de informações, personalização de clientes). O reuso do conhecimento nestas áreas gera economia de recursos e de tempo.
- **Facilidade nas manutenções preventiva, adaptativa e corretiva dos processos de negócio**: considerando que a semântica dos processos de negócio, por meio de regras, geralmente é espalhada em vários ambientes, então as regras armazenadas em um SGBDA tendem a ser menos complexas, requerendo menor quantidade de recursos para encaminhar uma solução, e como conseqüência a manutenção tende a ser menos complexa.
- **Centralização do conhecimento**: o intercâmbio de informações eletrônicas é, para muitas empresas, vantagem competitiva, visto que os processos de negócios estão automatizados. Um exemplo deste intercâmbio entre diferentes domínios são cadeias de supermercados e seus fornecedores. Centralizar estes processos de negócio em poucas aplicações, sob o domínio de uma mesma linguagem, em um mesmo ambiente computacional torna o processo de intercâmbio de informação mais simples, pois, exige menos recursos como redução no número de especialistas em cada uma destas aplicações, garantia de padronização de conceitos para a manutenção da lógica do negócio, além de custo financeiro e de tempo.
- **Entorno Cliente/Servidor**: é fato conhecido que os SGBDAs Relacionais são dominantes no mercado e que os ambientes de programação oferecem aos desenvolvedores a tecnologia Orientada a Objetos. Aglutinar estas duas tecnologias é um desafio presente em um ambiente de desenvolvimento, e que pode apresentar grandes dificuldades quando se

trata de manipular grandes quantidades de informação, quando as regras do negócio se encontram na aplicação, e os dados, em bancos de dados. Considerando que em um SBDA, dados e regras compartilham o mesmo ambiente, então, é possível minimizar a impedância gerada pelos dois ambientes, aplicação e banco de dados, quando na aplicação são especificadas as solicitações e no banco de dados a execução destas solicitações. Além disso, descongestionam-se as redes quanto ao tráfego de informações, quando o processamento é realizado localmente.

- Além destes aspectos positivos, soma-se a versatilidade de um SGBDA em função de que existe um ambiente para o tratamento da segurança quanto ao acesso da informação nele contida como também acesso ao próprio banco de dados, mecanismos disponíveis para gerenciamento e controle da integridade dos dados (restrições de dados).

Apesar dos aspectos positivos, ainda é muito freqüente encontrar sistemas com implementação de regras em meio a seu código, fora do SGBDA. Esta decisão leva a situações nas quais regras se tornam **redundantes**, devido a vários fatores, como por exemplo, a falta de documentação atualizada, sobre a especificação da regra, e quando a manutenção for realizada por diferentes projetistas pode levar a diferentes interpretações do processo de negócio. As regras podem ser quase **inacessíveis**, visto que elas podem participar de complexos fragmentos de códigos, portanto ocultas no meio de milhares de linhas de código, em lógicas complexas, dificultando a sua identificação. As regras podem estar **perdidas**, ou seja, sistemas são modificados por diferentes desenvolvedores, e muitas vezes, ao não reaproveitar o código, por razões de desconhecimento da linguagem ou falta de conhecimento do processo de negócio, isola o fragmento de código analisado e o refaz segundo o seu entendimento.

Na literatura existem basicamente duas abordagens, desenvolvidas em paralelo, para a **gerência de regras** (CERI; WIDOM, 1990; CERI; MANTHEY, 1993; BERNDTSSON, 1994; MORIARTY, 1998; DIAZ; JAIME, 2000; CERI; COCHRANE; WIDOM, 2000; ROSS, 2003). Uma delas é sob a ótica de análise de negócios e a outra sob a ótica de Banco de Dados. Estas técnicas são totalmente independentes e desconexas. A gerência de regras trata da criação, eliminação e alteração de regras em SGBDA.

Os trabalhos desenvolvidos sob á ótica de análise de negócios têm como objetivo facilitar a atividade de gerência de regras para os analistas de negócios (VON HALLE, 2002; VASILECAS; LEBEDYS, 2006) e, portanto, as ferramentas propostas são totalmente

orientadas para usuários não técnicos, utilizando um vocabulário apropriado para os mesmos. Nesta abordagem propõe-se o uso de um software *front-end* como sistema de regras de um negócio (ROSS, 2003). Este sistema de regras é composto por um repositório central de regras, que as armazena em uma linguagem acessível pelos analistas de negócios (linguagem quase-natural), e uma máquina de regras, que consiste em um conjunto de programas especializados na gerência de regras. Atualmente, existem vários produtos comerciais que implementam esta abordagem, tais como, *ILO*G (ILOG RULES, 2005), *Rule Track* (BUSINESS RULE GROUP, 2000) e *Business Rule Studio* (RULEMACHINES, 2000). A linguagem de definição de regras varia de acordo com o produto utilizado; a única semelhança está em que é uma linguagem muito próxima à linguagem natural. Uma das vantagens desta abordagem é que os profissionais da área de negócio podem definir e gerenciar as regras por meio de uma interface simples, melhorando assim a comunicação entre analistas de negócio e profissionais técnicos. Porém, estes sistemas somente abrangem as regras mais simples, não considerando as regras mais complexas embutidas dentro do código dos programas de aplicação. Outra desvantagem destes sistemas é a ineficiência na gerência de grandes quantidades de regras.

A outra abordagem, que analisa as regras sob a ótica de Banco de Dados, sugere aproveitar a infra-estrutura do próprio SGBDA para a gerência de regras. As regras são armazenadas em estruturas, assim como os dados, e o próprio SGBDA é utilizado como sistema gerenciador de regras. Esta abordagem é adotada neste trabalho.

Uma das vantagens desta abordagem é a possibilidade de gerenciar as regras e os dados em um único ambiente, simplificando assim a atividade de gerência desses objetos do banco de dados. As regras são definidas uma única vez no banco de dados e podem ser compartilhadas entre várias aplicações, existindo um controle centralizado das regras. Além disso, vários trabalhos têm sido propostos na literatura para auxiliar o projeto de regras em sistemas de informação, tais como navegadores de regras (LANG, 1998), depuradores de regras (KAPPEL; KRAMLER; RETSCHITZEGGER, 2001; ELIZONDO, 1998), analisadores de regras (VADUVA, 1999). Um navegador de regra permite a inspeção de um conjunto de regras existentes em um repositório de regras. Por meio deste navegador é possível criar, eliminar e alterar regras, como também é possível saber quais regras foram disparadas por uma determinada regra. Um depurador de regras é uma ferramenta que permite controlar a execução das regras com a finalidade de verificar se o repositório de regras implementa o comportamento requerido, pelo usuário, adequadamente. Um analisador de

regras é uma ferramenta que permite verificar se existe inconsistência no conjunto de regras, por exemplo, se não ocorrem ciclos infinitos quando um conjunto de regras é disparado.

No entanto, a maioria das organizações que opta por especificar as regras de negócio nos bancos de dados, enfrenta grandes dificuldades para o gerenciamento dessas regras em seus sistemas de informação, pois falta uma infra-estrutura apropriada para realizar as atividades de gerência de regras, de maneira ágil e eficiente. Esta infra-estrutura representa um conjunto de tabelas necessárias para armazenar as regras e um conjunto de mecanismos para garantir que as regras armazenadas nestas tabelas mantenham a devida consistência. Este mecanismo é representado por um conjunto de regras de consistência e estruturas que atuam de forma a garantir o emprego destas regras quando solicitadas. A atividade de gerência tem como objetivo fornecer recursos para definir, eliminar, alterar e consultar regras ou partes destas.

Assim como os dados, as regras também necessitam contar com um mecanismo de gerência eficiente, pois na medida em que se apresentam mudanças no negócio, surge a necessidade de mudanças também nas regras. Tais mudanças se refletem por meio da criação de novas instâncias de regras, eliminação de algumas regras já existentes ou a eliminação e adição de determinadas partes delas. Atualmente, **não é possível alterar partes das regras**, pois nos SGBDs atuais elas são tratadas como um elemento único. Esta incapacidade ocorre devido a dois fatores: o primeiro em decorrência de que os repositórios de regras não estão preparados para tratar alterações de partes de regras, e o segundo fator está em que não existem operações de gerência definidas para a alteração de partes de uma regra.

Devido à evolução dos processos de negócio, existe a necessidade de armazenar as regras em estruturas apropriadas para facilitar o seu gerenciamento, assim como, existem estruturas próprias para o armazenamento dos dados no banco de dados. Tecnicamente, fazendo um comparativo com os dados, isso significa que os dados são definidos de forma independente aos outros objetos de banco de dados, por meio de uma linguagem de definição de dados, e podem ser consultados ou atualizados por meio de uma linguagem de manipulação de dados. No entanto, uma regra, somente, pode ser criada ou eliminada, não existindo a possibilidade de realizar operações de alteração sobre ela. Além disso, no dicionário de dados, são armazenados os relacionamentos das regras com os dados, **deixando de lado as informações referentes aos relacionamentos entre as regras**. Esta informação sobre o relacionamento, entre regras, é importante porque reflete a própria lógica do negócio e também, representa o próprio encadeamento entre elas. Portanto, uma lógica de negócio pode

ser representada por um encadeamento de regras, porém, nem todo encadeamento representa uma lógica de negócio, como por exemplo, podem ser encadeamentos ad hoc que levam a comportamentos indesejados.

Com estruturas apropriadas, atuando como repositório de regras é possível armazenar partes das regras, o relacionamento entre regras e delas com os demais objetos do banco de dados. As regras de um sistema se relacionam entre si, da mesma forma em que os dados estão inter-relacionados. Elas também se relacionam com os dados, visto que muitas regras expressam restrições sobre eles, gerando assim uma complexa rede de associações, composta de associações entre regras e restrições destas sobre os dados. Estes aspectos semânticos, informações sobre a estrutura de uma regra e seus relacionamentos são a base para elaborar um ambiente que suporte operações de gerência de regras, e como conseqüência, dispor de um sistema de regras mais flexível em comparação com os sistemas atualmente oferecidos.

A linguagem SQL3 sugere um conjunto de operações de gerência, tanto para dados quanto para regras. No caso dos dados, existe um conjunto de operações para a definição e manipulação deles. A linguagem de definição de dados (LDD) especifica como criar as estruturas que vão comportar os dados da aplicação (por exemplo, as tabelas) em conjunto com mecanismos que auxiliam sua organização interna (por exemplo, a criação de índices). A linguagem de manipulação de dados (LMD) especifica a maneira como manipulá-los por meio de operações de gerência, tais como inserção, exclusão, consulta e atualização de dados (FORTIER, 1999). Na linguagem SQL3, as operações para definição e exclusão de uma regra estão embutidas com as operações de definição dos dados e, além disso, estas regras são **incluídas junto ao dicionário de dados** do banco de dados (ISO/IEC 9075-2, 1999). As regras não têm um tratamento independente dos dados, pois, ao criar uma regra, é necessário recorrer à LDD para encontrar a sintaxe de criação de uma regra. Neste sentido, o SGBDA não tem sido muito aproveitado para a implementação e gerência de regras.

Para que as regras alcancem o mesmo grau de importância que os dados em um SGBDA, é necessário que elas tenham um tratamento independente dos dados, apoiado por uma linguagem de gerência de regras e um repositório de regras. Entretanto, existem poucos trabalhos que tratam sobre a gerência de regras em SGBDAs (DIAZ; PATON; GRAY, 1991; CERI; MANTHEY, 1993; BERNDTSSON, 1994; MORIARTY, 1998; CERI; COCHRANE; WIDOM, 2000; ROSS, 2003) apesar da relevância desta tarefa.

Apesar das vantagens em implementar regras em um SGBDAs, ainda existe a necessidade de estender o modelo de regras da SQL3, pois este modelo de regras não é

suficiente para representar a grande maioria dos tipos de regras utilizada para especificar regras de negócio (PAVON; VIANA; CAMPOS, 2006). Um modelo de regras compreende um modelo de definição e um modelo de execução de regras. O modelo de definição apresentado pela SQL3 é a própria sintaxe da regra ECA, e o modelo de execução é definido pelo modo como este tipo de regra é executado. Para se estender um modelo de regras, por exemplo, o modelo apresentado pela linguagem SQL3, é necessário identificar e especificar as características das regras que se pretende estender. Isto pode ser feito por meio de um meta-modelo de regras. Existem vários trabalhos que propõem meta-modelos de regras como extensão da linguagem SQL3 (AMGHAR; MEZIANE; FLORY, 2000; TORRICO; TANAKA; MOURA, 2000), porém nem todos apresentam uma solução mais abrangente, que envolve tanto o meta-modelo quanto o modelo de regras (PAVON, 2005).

O modelo proposto em Pavón estende o modelo de regras apresentado pela SQL3. Este modelo é composto por um conjunto, mais amplo, de tipos de regras que os tipos apresentados pela linguagem SQL3. Nesta linguagem SQL3, pode-se representar regras do tipo ECA (evento – condição - ação) e, na omissão da condição, do tipo EA (evento - ação). No modelo proposto, o conjunto de tipos de regras envolve, além de ECA e EA, os tipos ECAA (evento - condição - ação primária - ação secundaria), CAA (condição - ação primária - ação secundária), CA (condição - ação[1]) e A (ação). Cada tipo de regra representa uma semântica diferente. Esta melhora na expressividade da regra implica que é possível expressar um conjunto de informações, mais amplo.

Além de ampliar o conjunto de tipos de regras, a linguagem proposta define um conjunto mais significativo de informações que podem ser especificados para cada elemento da regra. Por exemplo, o evento na linguagem SQL3 pode ser definido em função de três operações de manipulação de dados, a saber: uma inserção, uma atualização e uma exclusão de dados. Na linguagem proposta, este conjunto é ampliado, e considera além das operações descritas, outros tipos de operações que envolvem a seleção de dados, acesso ao banco de dados, entre outras. No modelo de regras proposto, portanto, é semanticamente mais rico que o modelo de regra da linguagem SQL3 e, portanto é considerado como ponto de partida para a proposta de um conjunto de operações de gerência de regras.

Considerando as desvantagens da linguagem SQL3, e os esforços em melhorar a expressividade desta linguagem, especificamente no que se refere às operações de gerência de

[1] O termo ação e ação primária têm o mesmo significado. Usa-se ação primária somente em presença de uma ação secundária.

regras, é possível concluir que ainda existe a necessidade de contar com uma infra-estrutura propícia para armazenar e gerenciar os principais tipos de regras encontrados nos sistemas de informação. Portanto, esta infra-estrutura tem como premissa a extensão proposta de tal forma que nela seja possível armazenar os relacionamentos existentes entre os diferentes tipos de regras e delas com os demais objetos do banco de dados. Esta infra-estrutura deve contar com um repositório de regras, para armazenar estes novos tipos de regras, com operações de gerência para atuar sobre as regras armazenadas neste repositório e um conjunto de regras de consistência para garantir a consistência do repositório de regras. Este trabalho de tese tem como finalidade de estender a linguagem de gerência de regras sugerida pelo modelo adotado, ou seja, oferecer um conjunto de operações de gerência, além das operações que são sugeridas neste modelo.

1.2 OBJETIVO

Considerando que as regras são tão importantes quanto os dados, existe a necessidade de contar com um tratamento para regras, da mesma forma que existe para os dados em um SGBDA, atribuindo-se o mesmo grau de importância a ambos. Para isto é necessário definir estruturas apropriadas para o armazenamento e organização dos tipos de regras sugeridos pela linguagem SQL3 e os tipos de regras apresentadas na literatura, como extensão desta linguagem. Estas estruturas devem prover um contexto que facilite a definição e a alteração de regras, e de suas partes. Como conseqüência, o ambiente apresentado deve ser propício para a gerência de regras, similar ao ambiente existente nos sistemas de bancos de dados, para a definição e manipulação de dados.

Este trabalho tem por objetivo geral especificar uma Linguagem de Gerência de Regras para Sistemas de Banco de Dados Ativos que permita gerenciar um sistema de regras, tendo como referência as operações de gerência sugeridas pela linguagem SQL3. Essa Linguagem de Gerência de Regras é proposta a partir de um modelo já proposto na literatura (PAVON, 2005) que estende a expressividade da linguagem de regras da SQL3, quanto ao seu modelo de definição de regras. Tal modelo de definição é adotado como referência para a elaboração de um conjunto de operações de gerência de regras. Neste trabalho não são considerados nem analisados aspectos relativos ao desempenho da linguagem, mas sim, a expressividade da linguagem proposta frente a linguagem de regras sugerida pela linguagem SQL3.

Os objetivos específicos, utilizados como marcos para se chegar ao objetivo geral são os seguintes.

- Fazer uma análise descritiva do modelo de regras adotado. Esta análise visa identificar as características dos tipos de regras que serão utilizados, sendo que estas características servem de base para a elaboração do repositório de regras.
- Estabelecer um repositório de regras capaz de armazenar os diferentes tipos de regras. Este repositório deve contemplar estruturas para o armazenamento das partes das regras e as associações entre elas.
- Identificar e descrever as meta-regras que devem ser adotadas para garantir a integridade do repositório de regras.
- Estabelecer as operações de gerência de regras. Para cada operação é necessário avaliar a sua conseqüência perante o repositório de regras.
- Identificar e descrever as meta-regras que devem ser adotadas para garantir o correto uso das operações de gerência.
- Elaborar um estudo de caso que servirá como ambiente de teste para as operações de gerência.

1.3 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 apresenta-se uma revisão da literatura, descrevendo-se os principais conceitos sobre banco de dados ativos, destacando-se os mecanismos utilizados por estes sistemas gerenciadores para a representação de regras, em especial, o modelo de regra ECA. No capítulo 3 é descrito o meta-modelo de regras, que sintetiza as propriedades das regras que formam a base para a proposta de extensão da linguagem de regras, proposta para a linguagem SQL3. Esta extensão inclui um modelo de regras, composto pelo modelo de definição e o modelo de execução de regras. Neste capítulo, é descrito o modelo de definição de regras, visto que este forma a base para a elaboração das operações de gerência de regras. Além disso, são apresentadas algumas operações de gerência de regras disponíveis na literatura e também as operações de gerência sugeridas no modelo adotado.

No capítulo 4, são apresentadas as estruturas utilizadas para o armazenamento dos tipos de regras que foram propostas no meta-modelo. Também é elaborado um conjunto de regras necessárias para garantir a consistência do repositório de regras. Este capítulo é concluído com um exemplo da aplicabilidade deste repositório.

No capítulo 5 é especificado um conjunto de operações de gerência de regras. Esta proposta inclui um conjunto de operações para a criação, exclusão e alteração de regas e de suas partes. Esta proposta, de uma linguagem de gerência, considera os mecanismos atualmente existentes na linguagem SQL3 e aproveita a extensão proposta para a linguagem de definição de regras. É apresentado um comparativo entre a linguagem de regras SQL3 para a gerência de regras e o conjunto de operações de gerência sugerido neste trabalho.

No capítulo 6 é apresentado um estudo de caso para avaliar a proposta do repositório de regras e dos mecanismos de gerência utilizados. São elaborados dois processos de negócio, que por sua vez são armazenados no repositório de regras e logo a seguir são utilizadas operações de gerência de regras sobre eles, segundo critérios previamente estabelecidos.

Por último, no capítulo 7, apresentam-se as conclusões finais deste trabalho, as principais contribuições e algumas sugestões para pesquisas futuras.

CAPITULO 2. REGRAS EM BANCO DE DADOS ATIVOS

2.1 INTRODUÇÃO

Neste capítulo são apresentadas, uma revisão do estado da arte sobre regras em Sistemas Gerenciadores de Banco de Dados Ativos (SGBDAs) e os mecanismos utilizados nestes sistemas para a representação de regras. São apresentados os modelos de regras sugeridos pela **linguagem SQL3**, os modelos utilizados pelos SGBDAs comerciais para a especificação de regras, e as estruturas utilizadas por esses gerenciadores para armazenamento dessas regras. Além disso, apresentam-se as limitações dos repositórios destes SGBDAs para o armazenamento de regras. Da mesma forma, são apresentadas as operações de gerência, sugeridas pela linguagem SQL3 e utilizadas pelos SGBDAs, com a finalidade de identificar as suas limitações, em decorrência destes repositórios. Por último, conclui-se com uma breve discussão sobre os problemas atuais referentes ao armazenamento e gerência de regras em SGBDAs.

2.2 REVISÃO DO ESTADO DA ARTE SOBRE REGRAS EM SGBDA

No inicio da década de oitenta surgiu à expressão “sistemas de banco de dados ativos” (MORGENSTERN, 1983; STONEBRAKER; WOODFILL; ANDERSEN, 1983), cuja idéia central era de que SBD, deveriam responder de maneira inteligente e não trivial às solicitações dos usuários, por meio de suas diversas interações, e de forma inteligente tratar as conseqüências e implicações dessas interações. Estes sistemas constituídos de diferentes ambientes de trabalho tinham, em comum, a necessidade de organizar e manter enormes quantidades de informações inter-relacionadas. Estas informações eram constituídas de mensagens, arquivos, dados estruturados e programas. A este novo paradigma, atribuía-se o nome de Banco de Dados Ativos.

Foi a partir da década de noventa que os trabalhos científicos, na área de Banco de Dados Ativos, tornaram-se mais freqüentes. O contexto de ativo na tecnologia de banco de dados engloba tanto **regras dedutivas**, quanto **regras ativas**. No primeiro caso, a tecnologia de banco de dados dedutiva (GUERRINI, 1998) foi influenciada por trabalhos em programação lógica, sendo que neste caso, as regras dedutivas expressam conhecimento declarativo e têm sido utilizadas em aplicações como análises financeiras, suporte à decisão e

modelagem científica. No segundo caso, a tecnologia de banco de dados ativos foi influenciada pela inteligência artificial e implementa reações computacionais automáticas em resposta a eventos que ocorrem tanto interna, quanto externamente ao banco de dados (CERI; RAMAKRISHNAN, 1996). Neste trabalho, é dada ênfase ao segundo caso, em função de que a grande maioria das aplicações comerciais usa regras ativas.

As regras em bancos de dados ativos passaram a serem reconhecidas como elementos fundamentais que devem ser considerados, tanto pela comunidade científica, quanto para o entorno comercial, como um meio para a busca de soluções de problemas reais. Para a comunidade científica, os Bancos de Dados Ativos receberam atenção em vários segmentos como a especificação de diversas arquiteturas e o desenvolvimento de sistemas e de linguagens (PATON; DIAZ, 1999). No meio comercial, fabricantes de Sistemas Gerenciadores de Banco de Dados procuraram incorporar regras (*triggers*) e a linguagem SQL3, sugerida pela **ISO / ANSI** (ISO/IEC 9075-2, 1999) passou a adotar um modelo de regras (PATON, 1998; FORTIER, 1999; TÜRKER, 2001). Um modelo de regras compreende um modelo de definição (sintaxe da regra) e seu modelo de execução de regras. O modelo de execução especifica como um conjunto de regras se comporta em tempo de execução, como por exemplo, sua política de prioridade de execução ou sua granularidade, entre outros.

Devido à riqueza de informações associadas a um Sistema Gerenciador de Banco de Dados Ativo (SGBDA), em 1996, Dittrich et al (ACT-NET CONSORTIUM, 1996) apresentaram um manifesto sobre os principais conceitos associados a esta área, sugerindo a uniformização de sua terminologia e apresentando as características obrigatórias e desejáveis que um SGBDA devesse conter. Alguns destes conceitos (PATON, 1998) são implementados em diversas arquiteturas de sistemas de banco de dados ativos, como NAOS (COLLET; COUPAYE; SVENSEN, 1994), SAMOS (GEPPERT et al., 1995a; GATZIU, 1997), ACOOD (ENGSTRÖM; BERNDTSSON; LINGS, 1997), ARIEL (HANSON, 1996), STARBURST (WIDON, 1996), ORACLE (ORACLE CORPORATION, 2003).

Existem diferentes abordagens para a especificação de uma arquitetura baseada em regras em um SGBDA. A primeira abordagem é a construção de um SGBDA a partir de sua concepção. A segunda abordagem é utilizar um SGBD passivo, modificá-lo e estendê-lo internamente. Os SGBDs convencionais são considerados passivos, porque executam operações de consulta ou transações sobre o Banco de Dados somente por solicitação explícita do usuário ou programa de aplicação. No entanto, os SGBDAs estendem os SGBDs passivos (PATON, 1999), incorporando a capacidade de executar automaticamente ações em resposta a

eventos gerados dentro ou fora do SGBDA, o que é possível por meio da especificação de regras ativas. Nesta segunda abordagem, o código fonte do SGBD deve estar disponível para que modificações possam ser realizadas, como por exemplo, SENTINEL (CHAKRAVARTHY et al., 1994), POSTGRES (STONEBRAKER; HEARST; POTAMIANOS, 1989; STONEBRAKER, 1992;), STARBURST (WIDON, 1996). A terceira abordagem é implementar, externamente, as funcionalidades ativas sobre um SGBD passivo. Estas funcionalidades fazem parte de um *framework* que utiliza as funcionalidades disponíveis do SGBD passivo. Esta estratégia é utilizada pelos Sistemas ACOOD[2], SAMOS[3] e TriGS (KAPPEL; RETSCHITZEGGER, 1998).

As duas últimas abordagens, aproveitam o dicionário de dados como um meio para o armazenamento de regras e o fazem, estendendo-o, ou seja, especificando e incorporando novas estruturas para o armazenamento de regras, junto ao dicionário de dados. Este mesmo conceito é apresentado pela norma que define a linguagem SQL3 (ISO/IEC 9075-2, 1999), pela qual o dicionário de dados contém as estruturas utilizadas para o armazenamento de *triggers*. A linguagem SQL3 não define claramente uma infra-estrutura voltada para regras, tendo como base um repositório de regras, da mesma forma que existe tal infra-estrutura para os dados, em um dicionário de dados. Com uma infra-estrutura para regras, seria possível promover as regras ao mesmo nível dos dados, assim como os dados o são em um SGBDA. Existem propostas de repositório de regras voltadas para o modelo relacional (HERBST; MYRACH, 1995; BUTLERIS; KAPOCIUS, 2002), e para o modelo orientado a objetos (GEPPERT et al., 1995a; DITTRICH et al., 2000). As propostas para o modelo relacional sugerem um repositório de regras externo ao SGBDA. Este trabalho segue a abordagem relacional porque é um modelo consolidado e amplamente utilizado na indústria.

Os repositórios de regras são a base para a especificação das operações de gerência de regras, pois sobre estas estruturas são realizadas inserções, modificações, exclusões e consultas às regras. Existem poucos estudos, apesar de sua importância, especificamente sobre a gerência de regras em SGBDA (CERI; MANTHEY, 1993; BERNDTSSON, 1994; WIDOM, 1996; CERI; COCHRANE; WIDOM, 2000; BONATTI et al., 2004), apesar de que, há preocupações sobre o comportamento das regras e como prover assistência a elas. Neste sentido, existem propostas de ferramentas para a assistência às bases de regras, como analisadores de regras (GUISHENG et al., 1996; BARALIS; CERI; PARABOSCHI, 1998;

[2] Active Object-Oriented Database System
[2] **S**wiss **A**ctive **M**echanism-Based **O**bject-Oriented Database **S**ystem

VADUVA, 1999; BAILEY; POULOVASSILIS; NEWSON, 2000), depuradores de regras (CHAKRAVARTHY; TAMIZUDDIN; ZHOU, 1995; DIAZ; JAIME, 2000; KAPPEL; KRAMLER; RETSCHITZEGGER, 2001) e editores de regras (FORS, 1995; LANG, 1998).

Existem produtos comerciais dedicados ao armazenamento e gerência de regras, cada qual com sua própria linguagem de regras, como ILOG (ILOG RULES, 2006) e Rule Track (BUSINESS RULE GROUP, 2000) acompanhados de suas próprias estruturas de armazenamento. As regras especificadas nestes sistemas são muito próximas à linguagem natural. A desvantagem destes sistemas reside em que, consideram regras simples, omitindo regras complexas como cadeias de regras "*If ... Then*" ou sentenças do tipo "*case"*, freqüentemente utilizadas em linguagens de programação. Além disso, estes sistemas são ineficientes no gerenciamento de grandes quantidades de regras.

Nos últimos anos, repositórios para armazenamento de dados XML, implementados como *framework*, externos aos SGBDA, têm sido utilizados para armazenar processos de negócios (RODRIGUES; CORREIA, 2005; VANHATALO; KOEHLER; LEYMANN, 2006), como conseqüência dos trabalhos científicos direcionados à área de *Internet*, discutindo assuntos sobre a necessidade de regras ECA (Evento-Condição-Ação) na *Web* e como especificar, formalmente, eventos compostos no contexto de linguagens *Web* (BONATTI et al, 2004; BRY; ECKERT, 2006), além de tratar, formalmente, regras no contexto da linguagem XML (*eXtended Markup Language*) (BONIFATI; CERI; PARABOSCHI, 2001; ABITEBUL; BENJELLOUN; MILO, 2004), utilizadas para construir aplicações *Web*.

2.3 REGRAS EM SISTEMAS DE BANCOS DE DADOS ATIVOS

Nesta tese, considera-se um Sistema de Banco de Dados Ativo (SBDA) como um sistema que se constitui de um Sistema Gerenciador de Banco de Dados Convencional (SGBD) adicionado de um componente ativo e um banco de dados. Este componente ativo é representado por um conjunto de funcionalidades do SGBDA que garantem a reação deste sistema, quando submetido a eventos. Tal componente ativo permite a especificação e execução de regras do tipo Evento – Condição – Ação (ECA). Estas regras são armazenadas em estruturas próprias no banco de dados, denominadas repositório de regras, que formam parte do dicionário de dados. A linguagem SQL3 possui um modelo de regras que permite a especificação de regras ECA, também denominado de *triggers* (ISO/IEC 9075-2, 1999). Além

das regras ECA, existem outros mecanismos para especificação de regras que também são armazenadas no dicionário de dados. Estes mecanismos são restrições de integridade, também conhecidas como *constraints*, definidas sobre tabelas especificadas no banco de dados. Estes *constraints* são verificados toda vez que um objeto do tipo tabela é definido ou um registro é inserido no banco de dados, e fazem parte da infra-estrutura definida para manipulação e definição de dados no banco de dados.

2.3.1 Restrições de Integridade

A linguagem SQL3 oferece os seguintes mecanismos para definir as restrições de integridade (PAVON, 1996; FORTIER, 1999; TÜRQUER, 2001).

- *PRIMARY KEY*: define uma restrição de chave primária em uma tabela.
- *UNIQUE*: não permite que uma coluna de uma tabela contenha valores repetidos.
- *FOREIGN KEY*: define uma restrição de integridade referencial.
- *DEFAULT*: especifica um valor padrão.
- *NOT NULL*: não permite que uma coluna de uma tabela deixe de conter valores.
- *CHECK*: define uma restrição sobre os possíveis valores que podem ocorrer em uma coluna específica.

Uma restrição é identificada por um nome e uma expressão condicional, utilizada para especificar a restrição de integridade a ser verificada. A violação a esta restrição representa uma rejeição à atualização do banco de dados que ativou a restrição.

Estas restrições são freqüentemente utilizadas na especificação de um esquema de banco de dados, e possuem uma infra-estrutura consolidada para o seu armazenamento e gerência.

Basicamente, as operações de gerência para a especificação de uma tabela são: criação (*create*), alteração (*alter*) e exclusão (*drop*). Com relação aos elementos da tabela, é possível alterar, adicionar e excluir uma coluna e seus respectivos tipos de dados, atribuir (*set*) e excluir o seu valor *default*. As restrições de valor único e de integridade referencial podem ser adicionadas ou excluídas da estrutura de uma tabela. Estas operações de gerência são

agrupadas na Tabela 2.1.

Tabela 2.1 – Operações de gerência sobre elementos associados a uma tabela.

Objetos do Banco de Dados	**Operações de Gerência**
Tabela	Criar, Alterar, Excluir.
Elementos da Tabela	Alterar, Adicionar, Excluir.
Restrições de valor único, valor *default*	Atribuir, Excluir.
Integridade Referencial	Adicionar, Excluir.
Expressão condiciona (*check*)	Adicionar, Excluir.

Na Tabela 2.2 é apresentada a sintaxe das restrições de integridade permitidas pela linguagem SQL3

Tabela 2.2 – Sintaxe das restrições de integridade na linguagem SQL3

Restrições de Banco de Dados
CREATE TABLE <nome-tabela> (<elemento-da-tabela> [{<elemento-da-tabela>}...]) elemento-da-tabela = <definição-da-coluna> | <definição-da-restrição-da-tabela> definição-da-coluna = <nome-da-coluna> <tipo-de-dados> | [DEFAULT <valor-padrão>] definição-de-restrição-da-tabela = <restrição-de-valor-único> | <restrição-de-integridade-referencial> | CHECK <expressão-condicional> restrição-de-valor-único = UNIQUE | PRIMARY KEY <nome-coluna> [{nome-coluna}...] <restrição-de-integridade-referencial = FOREIGN KEY <nome-coluna> (<colunas-referenciadas>) <especificação-de-referências> colunas-referenciadas = < nome-coluna > [{nome-coluna}...] especificação-de-referências = REFERENCES <nome-tabela> < nome-coluna > [{nome-coluna}...]

2.3.2 Domínios e Asserções

Além destas restrições, existem restrições de domínio (*domain*) e restrições afirmativas (*assertion*):

- *DOMAIN*: define os possíveis valores que pode conter um atributo.

- *ASSERTION*: define restrições de integridade sobre os valores de uma tabela com base em considerações de busca.

A sintaxe para a especificação de um *Domain* é a seguinte:

CREATE DOMAIN <nome-dominio> <tipo-de-dados>
[DEFAULT <valor-padrão>]
[lista-de-restrições-do-domínio]

As operações de gerência associadas a esta sintaxe são: criação, alteração e exclusão de um domínio.

A sintaxe para a criação de uma *assertion* é a seguinte:

CREATE ASSERTION <nome-restrição>
CHECK <expressão-condicional>

As operações de gerência associadas a esta sintaxe são: criação e exclusão de uma afirmação.

2.3.3 Funções e Procedimentos

Existem também rotinas na linguagem SQL3, que podem ser utilizadas para a especificação de regras, estas rotinas são procedimentos ou funções armazenadas. Uma rotina é um bloco SQL nomeado e, uma vez criado, é armazenado no banco de dados. A diferença na especificação de uma função com relação a um procedimento está na adição da cláusula *Return* diante do nome da Função. A sintaxe utilizada para a especificação de um procedimento na linguagem SQL3 é (ISO/IEC 9075-2, 1999; TÜRQUER; GERTZ, 2001):

CREATE PROCEDURE <nome do procedimento>
({<argumento> <modo> <tipo-de-dados> <vírgula>} ...)
[AS]
<bloco SQL>

<nome do procedimento> : identifica o nome do procedimento

<argumento> : especifica se uma ou mais variáveis são passadas como parâmetros para o procedimento. Para cada variável, deve ser especificado um tipo de argumento (*IN*, *OUT* ou *IN OUT*), especificado em <modo> seguido do seu tipo de dado. Por exemplo, "nome_usuário *IN* varchar(20)" (variável *IN* tipo-de-dado). O tipo de argumento *IN* passa um valor externo ao procedimento para o procedimento, *OUT* realiza a operação inversa e *IN OUT* combina as duas operações.

As rotinas são armazenadas em tabelas próprias, e as operações de gerência associadas a elas são: criação, alteração e exclusão.

2.3.4 Triggers

Um outro objeto empregado para a representação de regras em sistemas de informação são os *triggers*. Estes mecanismos, ao contrário das rotinas e das restrições de integridade, têm a capacidade de reagir a diferentes eventos, internos ou externos ao banco de dados, da mesma forma que são utilizados para monitorar situações de interesse no banco de dados, e uma vez detectadas, executam-se ações previamente definidas. Uma característica dos *triggers*, quando comparados com as rotinas, é o fato de serem disparados, automaticamente, sem o recebimento ou retorno de parâmetros (ELMASRI; NAVATHE, 2005; FORTIER, 1999). A sintaxe de definição de um *trigger*, segundo a notação BNF, é a seguinte (ISO/IEC 9075-2, 1999).

CREATE TRIGGER <**nome-trigger**>
{BEFORE|AFTER} <**evento**>
[REFERENCING <variáveis de transição>]
[FOR EACH {ROW|STATEMENT}]
[when (<**condição**>)]
  <**ação**>

Cada *trigger* tem um nome que o identifica, um evento, uma condição e uma ação. Além destes três elementos, fazem parte do *trigger* informações sobre seu tempo de ativação e variáveis de transição. O tempo de ativação representa uma relação entre o momento de ocorrência do evento e o momento de execução da regra. A execução de uma regra representa a avaliação de sua condição e execução de sua ação. Existem dois tipos de ativação, antes (*before*) ou depois (*after*). Assim, uma regra pode ser executada antes da execução da operação definida como evento ou após a execução desta operação. A sintaxe da cláusula evento é a seguinte:

<evento> ::= {INSERT | UPDATE | DELETE} [OF <lista de colunas>]
ON <nome da tabela>

As operações de manipulação de dados (*insert*, *update* e *delete*) são utilizadas como evento em um *trigger*. Considerando a cláusula do evento, é possível especificar as colunas de interesse <lista de colunas> sempre que o evento for uma operação de atualização de dados (*update*) sobre alguma tabela. Desta forma, o *trigger* será disparado quando ocorrer alguma operação de atualização sobre as colunas listadas em sua definição. A cláusula <nome da tabela> refere-se à tabela sobre a qual é realizada a operação de atualização.

Na especificação do *trigger*, é possível fazer referência a valores passados e valores futuros que são atribuídos a um atributo, por meio de variáveis de transição. Os valores que são referenciados na variável *old* referem-se aos valores anteriores à modificação da tabela e os valores referenciados na variável *new* são os valores posteriores à modificação dela.

Na expressão <variáveis de transição>, da sintaxe do *trigger*, é possível especificar, também, nomes alternativos <nome do alias> para as variáveis de transição *old* e *new*, segundo a notação:

<variáveis de transição> ::= OLD [ROW] [AS] <nome do alias>
| OLD [TABLE] [AS] <nome do alias>
| NEW [ROW] [AS] <nome do alias>
| NEW [TABLE] [AS] <nome do alias>

Os elementos da sintaxe utilizada para especificar os valores de transição são descritos, a seguir, por meio de um exemplo ilustrativo:

Considerando a seguinte expressão: "*REFERENCING OLD* as valor_antigo". O

“valor_antigo” é utilizado para acessar os valores de referencia à variável *old*, independentemente que seja utilizada no elemento condição ou na ação do *trigger*. A variável guarda uma relação direta com a operação definida no evento do *trigger*. Quando a operação é de inserção pode-se utilizar somente a variável *new*, quando a operação é de atualização, as variáveis podem ser *old* ou *new* e para ocaso de que a operação seja uma exclusão então a variável deve ser *old.*

Na definição do *trigger*, também é especificada a granularidade de processamento da regra. A granularidade de processamento representa o número de vezes que o *trigger* é disparado em relação ao evento. Ela pode ser ao nível de tupla (*FOR EACH ROW*) ou ao nível de conjunto (*FOR EACH STATEMENT*). Quando não é especificada a granularidade da regra, assume-se a granularidade de conjunto.

Na <condição> do *trigger* pode ser especificado um predicado sobre o estado do banco de dados, na forma de uma expressão simples ou uma consulta SQL. A condição pode apresentar operadores booleanos binários (OR ou AND) com a finalidade de combinar outros predicados.

No seguinte exemplo, são empregadas variáveis de transição, granularidade da regra e uma condição simples.

```
CREATE TRIGGER atualizar_salario
AFTER UPDATE ON EMPREGADO
REFERENCING OLD as salario_antigo
                    NEW as salario_atual
FOR EACH ROW
WHEN (salario_antigo.salario < salario_atual.salario)
… <ação do trigger>
```

Neste exemplo, o *trigger* é denominado “atualizar_salario”, e tem especificada na cláusula de evento a definição de uma operação de banco de dados *UPDATE*, sobre a tabela EMPREGADO. Definem-se duas variáveis de transição, “salario_antigo” e “salário_atual”, como também é especificada a granularidade de processamento da regra (*FOR EACH ROW*). Portanto, para cada registro da tabela EMPREGADO em que o salário é atualizado, verifica-se a condição da regra (*WHEN*). No caso que seja verdadeira, a ação especificada na <ação do *trigger*> é executada, caso contrário, nada acontece.

Na <ação> de um *trigger*, é possível especificar uma sentença procedimental SQL ou um bloco SQL, segundo a especificação:

<ação> ::= <sentença SQL procedimental>
<bloco SQL>
<bloco SQL> ::= BEGIN
{<sentença SQL procedimental> <ponto e vírgula>} ...
END;

Uma sentença SQL procedimental é um conjunto de operações ou sentenças válidas na linguagem SQL3 que incorpora recursos de programação, como por exemplo, *IF...THEN...ELSE, REPEAT, WHILE*, sentenças de linguagem de manipulação de dados, sentenças de controle de fluxo, variáveis e cursores, entre outros.

Um bloco SQL é um conjunto destas sentenças, delimitado pelas cláusulas *BEGIN* e *END*. O exemplo a seguir apresenta um exemplo de bloco SQL especificado na ação de um *trigger*.

```
CREATE TRIGGER valor_max_emprestimo
AFTER INSERT ON CLIENTE
FOR EACH ROW
WHEN (new.categoria = 'A')
DECLARE
    V_codpedido PEDIDO_EMPR.codpedido %TYPE;
BEGIN
    SELECT codpedido INTO V_codpedido
    FROM PEDIDO_EMPR
    WHERE codcli = :new.codcli;
    UPDATE PEDIDO_EMPR
    SET valor_max_empr = new.salario_anual * 0,5
    WHERE codcli = :new.codcli
    AND codpedido = V_codpedido;
END;
```

Neste exemplo, o *trigger* "valor_max_emprestimo" possui um evento definido pela operação de banco de dados *INSERT* sobre a tabela "CLIENTE", com um tempo de ativação

AFTER. Portanto, este *trigger* é disparado toda vez que houver a inserção de um registro nesta tabela.

Na condição do *trigger* é avaliado o valor do atributo "categoria", se o valor do atributo em questão é "A", então, a ação é executada. A ação compreende o código definido entre os delimitadores *BEGIN* e *END*. A ação é composta de um bloco SQL, contendo duas operações de banco de dados. A primeira operação consiste em uma consulta sobre a tabela PEDIDO_EMPR, com a finalidade de identificar o código do pedido associado ao cliente que foi inserido na referida tabela, por meio do evento que disparou esta regra. A segunda operação é composta por uma atualização de dados sobre a tabela PEDIDO_EMPR. Esta operação reajusta o valor do empréstimo do cliente em função de seu salário anual. As operações do tipo inserção, exclusão e atualização podem ser o marco inicial para o disparo de outras regras. No *trigger* apresentado, existe uma operação de atualização que ao ser executada, pode disparar qualquer outro *trigger* que possua um evento definido com esta mesma operação de banco de dados sobre a mesma tabela a qual incide o evento. A este evento define-se como evento de ativação. A este disparo de regras implícito, dá-se o nome de encadeamento de regras ou encadeamento de *triggers*. Um aspecto não permitido é a existência de um auto-*trigger*, ou seja, um *trigger* que possui, em sua ação, o seu evento de ativação. Todas as informações sobre os *triggers* são armazenadas no dicionário de dados dos SGBDAs.

As operações de gerência sobre a definição de um *trigger*, segundo a linguagem SQL3, são *CREATE TRIGGER* e *DROP TRIGGER*. A primeira operação tem a função de criar um *trigger* e armazená-lo no dicionário de dados, enquanto que a segunda operação exclui o *trigger* do dicionário de dados (TÜRQUER, 2001). Os *triggers* são armazenados em tabelas que formam parte do **dicionário de dados** junto com os meta-dados e as tabelas utilizadas para armazenar dados da aplicação. Na linguagem SQL3 não existe o conceito de **repositório de regras**, ou seja, um conjunto de tabelas utilizadas para armazenar os *triggers*, sob o domínio de uma infra-estrutura dedicada ao gerenciamento destas regras.

2.4 MODELO DE REGRAS EM SBDAS

Um Sistema de Banco de Dados Ativos possui um modelo de regras (PATON; DÍAZ, 1999), composto por um modelo de definição e um modelo de execução de regras. O modelo de definição determina o que deve ser especificado na regra (sintaxe da regra), enquanto que o

modelo de execução determina como será o processamento das regras. O modelo de regras varia de um SGBDA comercial para outro, no entanto os conceitos básicos adotados por eles estão presentes no modelo de regras da linguagem SQL3. A finalidade de descrever o modelo de regras da linguagem SQL3 reside no fato de que este modelo guarda uma relação direta com o repositório de regras, pois o repositório armazena as características ou propriedades das regras. O repositório de regras é conseqüência do modelo de regras adotado. A seguir, são detalhados os modelos de definição e execução utilizados pela linguagem SQL3.

2.4.1 Modelo de Definição de Regras

O modelo de definição de regras de um SGBDA compreende um conjunto de sentenças representadas por meio de uma linguagem de definição de regras, cuja finalidade é a especificação de regras ECA (Evento-Condição-Ação). A sintaxe genérica para a definição de uma regra ECA é a seguinte.

DEFINE RULE <nome_regra>
ON <evento>
IF <condição>
DO <ação>

Este padrão de regra tem sido seguido por vários SGBDAs comerciais, com a denominação de *triggers*, e cada sintaxe definida em um SGBDA comercial pode sofrer pequenas modificações, e isto depende diretamente das características que foram adotadas pelo SGBDA para a especificação de uma regra. (PATON; DIAZ, 1999; VADUVA, 1999; ELMASRI; NAVATHE, 2005).

Cada regra deve ser definida com um nome único. O evento da regra especifica um acontecimento que dispara a regra. A condição da regra representa um predicado que deve ser avaliado. Quando o resultado desta avaliação é verdadeiro, então a ação é executada. Quando este resultado é falso, a regra não é executada.

- Evento

Um evento é definido como uma ocorrência atômica sobre o banco de dados (DIAZ, 1991). O evento é produzido por uma fonte, a saber, um sistema informático externo ao SGBDA, o próprio SGBDA, um hardware ou interação do próprio usuário com o banco de

dados. Um SGBDA detecta a ocorrência dos eventos e dá continuidade ao processo com o disparo das regras que pode por ele serem disparadas. Uma regra é disparada por um evento, mas um evento pode disparar mais de uma regra.

Os eventos podem ser primitivos ou compostos. Um evento primitivo ou simples, ocorre quando se está diante de uma única ocorrência, como por exemplo, uma operação de banco de dados ou um fato temporal (ACT-NET CONSORTIUM, 1996; JAIME, 1998). A combinação de eventos primitivos com operadores (AND, OR, SEQUENCE) deriva os eventos compostos, que por sua vez podem conter outros eventos compostos (BERNDTSSON; MELLIN; HOGBERG, 1999). Na Figura 1, são apresentados estes conceitos (CHAKRAVARTHY; MISHRA, 1993).

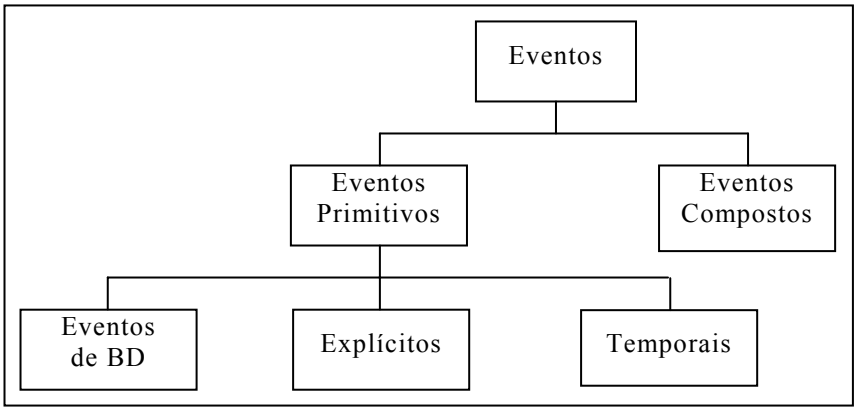

Figura 1 - Classificação de eventos nos SBDAs (CHAKRAVARTHY; MISHRA, 1993).

Esta classificação tem sido adotada por vários outros trabalhos (ACT-NET CONSORTIUM, 1996; GUERRINI, 1998; KANGSABANIK, 1998; VADUVA, 1999; BERNDTSSON; MELLIN; HOGBERG, 1999; TORRICO; TANAKA; MOURA, 2000; BONATTI et al., 2004; PAVON, 2005;), e apresenta os eventos primitivos como eventos de banco de dados, explícitos e temporais. Os eventos de banco de dados são operações de manipulação de dados (CHAKRAVARTHY; MISHRA, 1993) como *insert*, *update* e *delete*, considerando o modelo relacional. Quando o modelo é orientado a objetos, então as operações de eventos são mensagens ou chamadas a métodos (GEPPERT et al., 1995; GATZIU, 1997). Os eventos explícitos fazem parte da lógica do negócio que são especificadas na aplicação. Eventos temporais podem ser definidos dentro de dois domínios, absolutos ou relativos. O primeiro, evento temporal absoluto, diz respeito à definição de um momento específico no tempo, como por exemplo, “21 de Março de 2006 às 16 horas”. O segundo, evento temporal relativo, define uma ocorrência em relação à outra. “Uma hora após a ocorrência do primeiro

evento" ou um evento composto da união de dois eventos primitivos por um conector *booleano*.

Uma vez que o evento é detectado, ele é armazenado no repositório de dados, em uma tabela específica para armazenamento de suas informações, tais como o seu nome, o nome da tabela referenciada por ele e as colunas especificadas na sintaxe do evento. A existência de um repositório de regras é a base para que seja possível manipular as informações nele armazenadas. Porém, não existem operações de gerência sobre eventos. As operações de gerência de regras se limitam à sua criação e exclusão. Desta forma, para especificar um evento diferente para uma regra, é necessário eliminar e criar novamente a regra com o novo evento.

A regra pode ser disparada em função de dois momentos no tempo. O primeiro momento pode ser especificado por meio da sentença "antes" (*before*), sendo que a regra é disparada antes da ocorrência da operação do evento. O segundo momento é indicado por meio da sentença "depois" (*after*), no qual a regra é disparada depois da ocorrência da operação definida no evento. O evento é um componente obrigatório na especificação da linguagem de regras ECA (ACT-NET CONSORTIUM, 1996; TORRICO; TANAKA; MOURA, 2000).

- Condição

A condição é o componente que expressa o que deve ser avaliado para que a ação da regra seja executada. À semelhança do evento, a condição pode ser simples ou composta (HERBST; MYRACH, 1995). Uma condição simples representa um predicado atômico, definida por meio de uma expressão simples (por exemplo: maior que 18 anos) ou uma consulta SQL. Também é permitido o uso de variáveis de transição, *old* e *new*, nas consultas SQL, para referir-se aos valores antigos ou novos de um atributo. Quando a condição se compõe de vários predicados simples unidos por operadores *booleanos* do tipo *AND* ou *OR*, é dita composta.

As regras Evento-Ação (EA) são uma variação das regras ECA, tendo como característica a omissão da condição da regra (PATON; DIAZ, 1999). Desta forma, quando um evento é detectado, sua ação é imediatamente executada.

- Ação

A ação é um conjunto de operações que expressa a reação ao evento detectado. Na ação de um *trigger*, pode-se especificar um conjunto de sentenças procedimentais SQL ou um bloco SQL. Uma sentença procedimental pode ser uma sentença SQL ou uma sentença SQL estendida, que permite o uso de comandos de controle de fluxo, inclusive, o uso de variáveis de transição para acessar valores antigos ou valores novos de uma instância da tabela. Quando estas sentenças SQL envolvem operações de banco de dados, do tipo *insert*, *update* e *delete*, sobre algum objeto do banco de dados, e são também definidas em outras regras como eventos, então, este contexto forma um encadeamento de regras.

Problemas associados ao encadeamento de regras são amplamente discutidos na literatura (AIKEN; HELLERSTEIN; WIDOM, 1992; AIKEN; WIDOM; HELLERSTEIN; 1995; ZIMMER; UNLAND; MECKENSTOCK, 1996; VADUVA; GATZIU; DITTRICH, 1997; VADUVA, 1999; BARALIS; WIDOM, 2000; COUCHOT, 2001) não sendo, portanto, abordadas neste trabalho.

É possível utilizar, na ação, combinações de operações de banco de dado em uma sentença SQL, por meio de operadores *booleanos*, o que caracteriza uma ação composta. (TORRICO; TANAKA; MOURA, 2000)

2.4.2 Modelo de Execução de Regras

O modelo de execução de regras define como será o processamento de uma regra ou de um conjunto de regras (PATON; DIAZ, 1999; TURKER, 2001; VADUVA, 1999) e determina as propriedades da execução de regras, como granularidade de processamento e modo de acoplamento, entre outras. Com a finalidade de descrever melhor o alcance do modelo de execução, na Figura 2 são apresentados os principais passos realizados durante o processamento de um conjunto de regras (PATON; DIAZ, 1999).

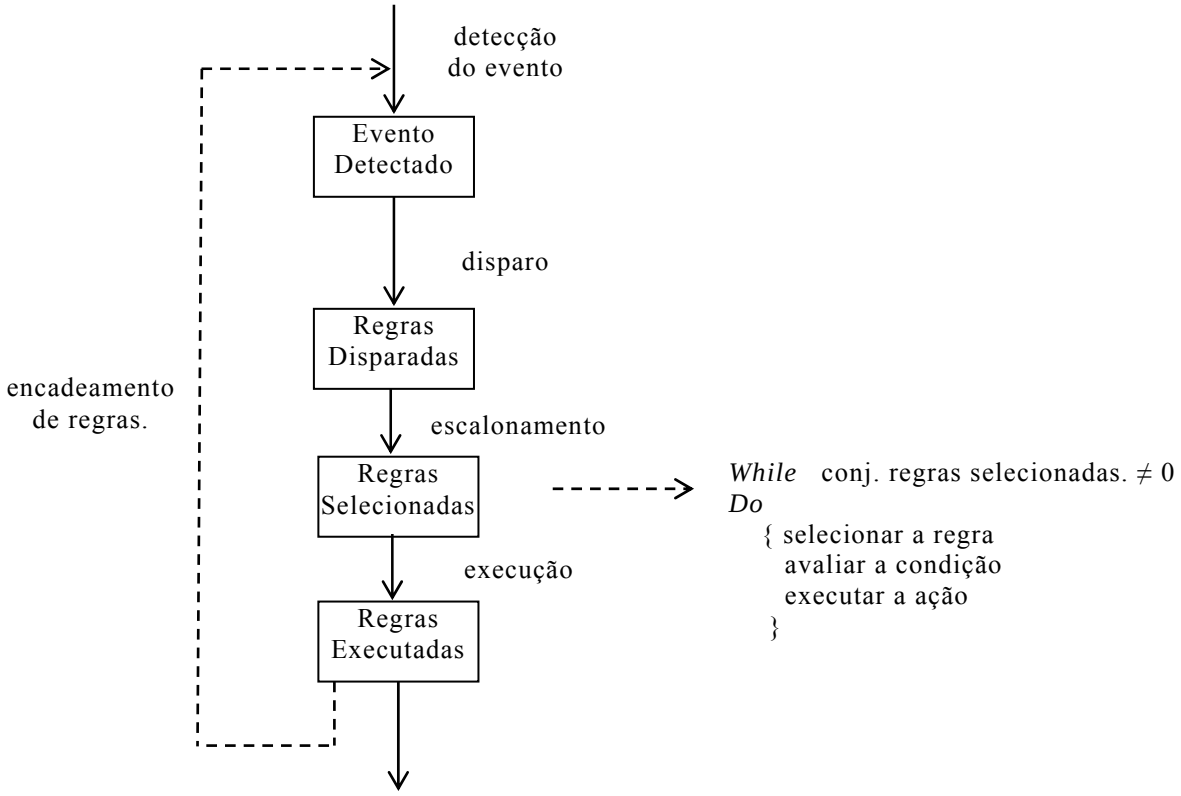

Figura 2 - Passos envolvidos no processamento de regras (PATON; DIAZ, 1999; VADUVA, 1999).

Sistemas de informação, freqüentemente, acessam bancos de dados para realizar modificação nos dados. Estas modificações são conseqüência do uso de operações de bancos de dados. Estas operações são detectadas pelo SGBDA, que por sua vez, dispara todas as regras que possuem como evento estas operações. O processamento de regra se inicia com a detecção do evento. Quando se trata de evento composto, um sistema de detecção de evento composto analisa as diversas ocorrências de eventos simples que contribuem para o evento composto desejado, e dispara as regras a ele associadas. Berndtsson apresenta uma análise de diversas técnicas para a detecção de eventos (BERNDTSSON; MELLIN; HOGNERG, 1999) que são utilizadas em diversos SGBDA, como SENTINEL (CHAKRAVARTHY et al., 1994) e SAMOS (DITTRICH et al., 2000).

Quando uma única regra é disparada, assume-se a detecção de um evento, avalia-se a condição e executa-se a ação, desde que a condição seja verdadeira. No entanto, em um sistema de regras, geralmente mais de uma regra é disparada, pelo mesmo evento, gerando um **conjunto de disparo simultâneo de regras**. Uma regra é "**disparada**" quando o seu evento é detectado pelo SGBDA, e a regra passa ao estado de "**execução**" quando a condição da regra

é avaliada e sua ação pode ser executada (PATON; DIAZ, 1999; PAVON, 2005). O processamento de consulta adotado considera a execução serial de regras, ou seja, uma regra por vez é disparada e executada. Porém existe outra visão, no qual regras são disparadas simultaneamente. Neste caso, a condição e a ação das regras são avaliadas e executadas concorrentemente, o que demanda um controle de concorrência de regras para que a execução de uma regra não interfira na execução da outra (KANGSABANIK, 1998; CERI; WIDOM, 1992).

Quando ocorre um conjunto de conflito de regras, está-se diante de um conflito quanto à ordem de execução das regras. Antes de aplicar um critério de seleção para a execução das regras, o SGBDA classifica as regras em dois grupos. As regras com tempo de ativação *before* e as regras com tempo de ativação *after*. O passo seguinte é executar as regras do primeiro grupo, e depois do segundo grupo. Considerando que existem várias regras, tanto do primeiro quanto do segundo grupo, então a **fase de escalonamento**, no processamento de regras, utiliza estratégias para solucionar este impasse por meio de critérios de prioridade ou precedência, com a finalidade de determinar a ordem de execução de tais regras.

Para alguns SGBDAs, o critério de prioridade é definido pela data/hora de criação da regra, como é o caso do SAMOS (PATON, 1998) e ORACLE (ORACLE CORPORATION, 2003). Existem outros SGBDAs que adotam pesos, definidos pelo usuário, e quando o peso é o mesmo para duas ou mais regras, então, o critério passa a ser aleatório. O processamento de regras pode se repetir quando uma regra possui, em sua ação, algum evento de ativação, que leve ao disparo de outras regras, reiniciando este processo. Um modelo para a resolução de conflitos é uma característica marcante dos SGBDAs, visto que, esta propriedade varia entre cada um deles. Portanto, um sistema de regras, ao ser especificado em um SGBDA, deve considerar esta propriedade.

No manifesto, ou documento de intenções, sobre regras em SGBDA (ACT-NET CONSORTIUM, 1996) é apresentado um conjunto de propriedades desejáveis que um modelo de execução de regras deve possuir, dentre estas, as principais são:

a) o SGBDA deveria implementar um modelo de resolução de conflito, por exemplo os que foram apresentados anteriormente;
b) Um SGBDA deveria atender ao conceito de granularidade de processamento de regras. Esta granularidade refere-se ao grau de relacionamento entre a ocorrência do evento e o disparo da regra. Existem basicamente duas dimensões para este relacionamento, a primeira é a granularidade de tupla, na qual a regra é disparada para

cada tupla afetada pelo evento, e a segunda, é a granularidade de conjunto, na qual a regra é disparada uma única vez para todo o conjunto de tuplas afetadas pelo evento;

c) Um SGBDA deveria oferecer um modo de acoplamento. O acoplamento ocorre entre os componentes evento-condição e os componentes condição-ação da regra. As regras são definidas dentro de transações, e os eventos fazem parte de transações também. No entanto, também existem modos de acoplamento que fazem referência a transações diferentes, para o mesmo processamento de regra. Estes acoplamentos são definidos como "desacoplados", pois a condição é avaliada dentro de uma transação diferente da transação no qual o evento ocorreu. Da mesma forma, a ação é executada dentro de uma transação diferente da transação na qual a condição ocorreu. Os acoplamentos referentes à mesma transação são definidos como: *imediato* e *atrasado*. O primeiro ocorre quando a condição é avaliada imediatamente depois que ocorreu o evento, ou a ação é avaliada imediatamente depois que a condição foi avaliada.

O segundo caso, *atrasado*, acontece quando a condição é avaliada dentro da mesma transação que o evento da regra, porém não necessariamente na primeira oportunidade, geralmente o processamento é adiado para o final da transação, ou a ação é executada dentro da mesma transação que a condição da regra, mas não necessariamente na primeira oportunidade. Em Paton (PATON; DIAZ, 1999) é apresentado um resumo dos principais SGBDAs com os seus tipos de acoplamento. A maioria deles opta pelo processamento da regra dentro da mesma transação.

2.5 ESTRUTURAS DE ARMAZENAMENTO DE REGRAS

O modelo de regras contém informações que devem ser atendidas pelo repositório de regras. Por exemplo, no repositório são armazenadas a data e hora de criação do *trigger*, os eventos definidos na sua sintaxe, o tempo de ativação deste evento, entre outros. Todo SGBDA possui um repositório de regras, para armazenar informações advindas do modelo de regras. Nesta seção, são analisados diferentes repositórios de dados: o recomendado pela linguagem SQL3 e um repositório comercial.

2.5.1 Repositório Sugerido pela Linguagem SQL3

O dicionário de dados proposto pela linguagem SQL3 (ISO/IEC 9075-2, 1999) armazena informações sobre definição das regras e o relacionamento delas com os meta-dados (tabelas e colunas). Na Figura 3 são apresentadas as estruturas sugeridas pela linguagem SQL3.

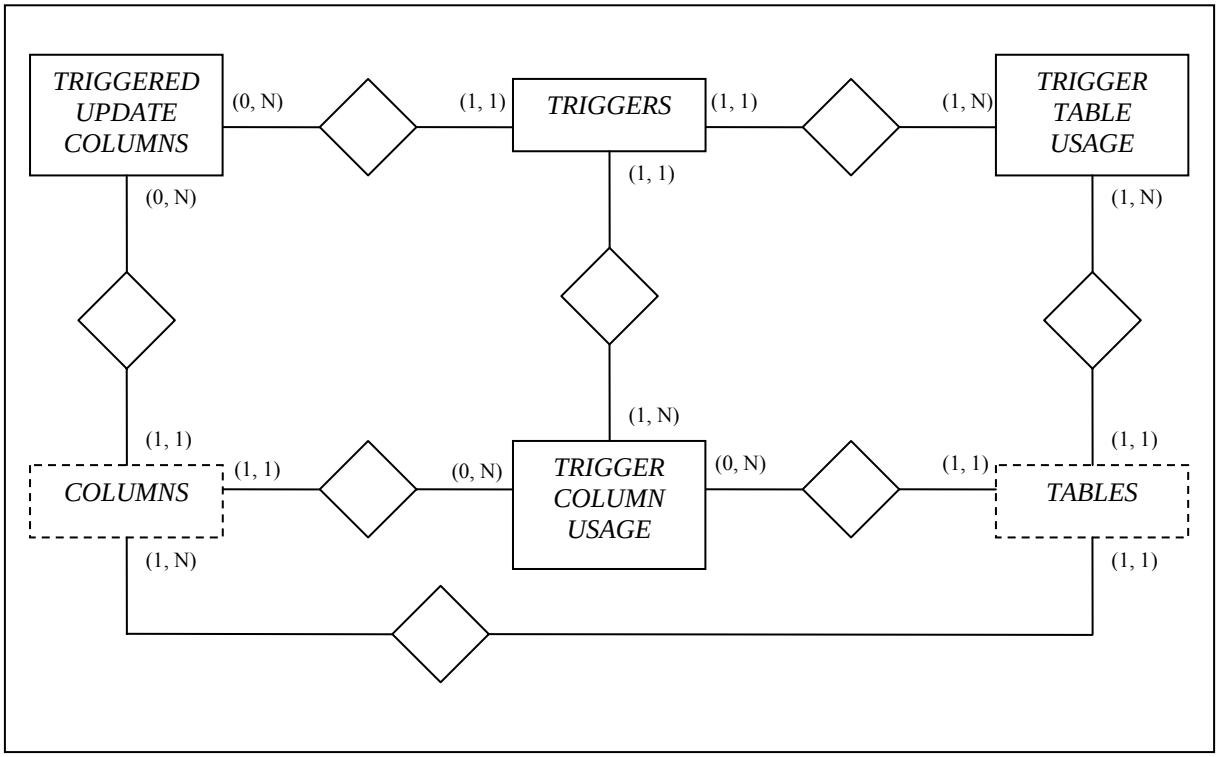

Figura 3 - Repositório de regras da linguagem SQL3 (ISO/IEC 9075-2, 1999)

A tabela *TRIGGER TABLE USAGE* armazena informações sobre tabelas utilizadas na condição ou na ação de um *trigger*. A tabela *TRIGGERS COLUMN USAGE* armazena informações sobre as colunas utilizadas no evento, na condição ou na ação do *trigger*, com exceção da operação de atualização que é especificada na tabela *TRIGGERED UPDATE COLUMNS*. A tabela *TRIGGERS* contém, efetivamente, todas as informações especificadas na definição do *trigger* como a definição da operação definida no evento, tempo de ativação, entre outros. Cada *trigger* deve estar associado a uma tabela, e pode estar associado, implícita ou explicitamente, a colunas desta tabela. Os meta-dados *TABLES* e *COLUMNS*, representados por retângulos pontilhados, têm por finalidade representar estas associações.

A Tabela 2.3 apresenta as informações mais relevantes sobre as tabelas que são

utilizadas em um *trigger* para efeito de análise neste trabalho. Por exemplo, as demais informações que constam da tabela *TRIGGER TABLE USAGE* e que não estão presentes nesta tabela são o nome do esquema do banco de dados ao qual pertence a tabela referenciada na condição e ação e o respectivo *trigger*. Nesta tabela, é inserido um registro para cada tabela referenciada na condição ou na ação do *trigger*.

Tabela 2.3 - Conteúdo da tabela *TRIGGER TABLE USAGE*

Coluna	Descrição da coluna
Trigger_name	Nome do *trigger*
Table_name	Nome da tabela referenciada na condição ou ação do *trigger*

A Tabela 2.4 apresenta as principais informações sobre as colunas utilizadas na condição e na ação do *trigger* Nesta tabela é inserido um registro para cada coluna referenciada na condição ou na ação do *trigger.* Além disso, também é armazenado o nome da tabela à qual pertence esta coluna e o nome do *trigger* correspondente. Assim como a tabela anterior, outras informações também fazem parte da tabela *TRIGGER COLUMN USAGE*, como o nome do esquema ao qual pertence à tabela e o *trigger*..

Tabela 2.4 Conteúdo da tabela *TRIGGER COLUMN USAGE*

Coluna	Descrição da coluna
Trigger_name	Nome do *trigger*
Table_name	Nome da tabela que contém a coluna referenciada no *trigger*
Column_name	Nome da coluna referenciada no *trigger*

Na Tabela 2.5 apresentam-se as principais informações sobre a tabela *TRIGGERED UPDATE COLUMNS*. Esta tabela armazena um registro para cada coluna contida na definição do evento, considerando que este evento seja uma operação *update*. As tabelas correspondentes a esta operação, também são armazenadas na tabela *TRIGGERED UPDATE COLUMNS*.

Tabela 2.5 - Conteúdo da tabela *TRIGGERED UPDATE COLUMNS*

Coluna	Descrição da Coluna
Trigger_name	Nome do *trigger*
Event_object_table	Nome da tabela afetada pela operação definida como evento
Event_object_column	Nome da coluna especificada na <lista-de-coluna> na definição do *trigger* (especificamente para operações *update*)

Na Tabela 2.6 apresentam-se as informações da tabela *TRIGGERS*. A tabela *TRIGGERS* contém um registro correspondente a cada *trigger* instanciado ou criado. Para cada um deles é atribuída uma única operação de banco de dados. O atributo *action_order* contém o valor da posição do *trigger* dentro de uma lista de *triggers* definidos com o mesmo evento sobre a mesma tabela, e portanto com o mesmo tempo de ativação e com a mesma granularidade. Outros atributos que fazem parte da tabela, tais como o nome do esquema de banco de dados a que pertence o *trigger*, os valores de transição (*old* e *new*) não foram adicionados pelo fato de que não agrega valor a análise que é feita sobre o repositório.

Tabela 2.6 - Conteúdo da tabela *TRIGGERS*

Coluna	Descrição da Coluna
Trigger_name	Nome do *trigger*
Event_manipulation	Operação de BD definida como evento (*insert*, *update* e *delete*)
Event_object_table	Nome da tabela afetada pela operação definida no evento
Action_order	Posição do *trigger* dentro do conjunto de conflito
Condition	Condição do *trigger*
Action	Ação do *trigger*
Action_Orientation	Granularidade de processamento do *trigger* (*row* ou *statement*)
Condition_time	Tempo de ativação do *trigger* (*before* ou *after*)
Timestamp	Data e hora de criação do *trigger*

As estruturas apresentadas no dicionário de dados da linguagem SQL3, para armazenar regras, apresentam dois grupos de informações bem definidas. O primeiro grupo, formado pelas tabelas *TRIGGER TABLE USAGE, TRIGGERS COLUMN USAGE*, e *TRIGGERED UPDATE COLUMNS*, armazena informações sobre as tabelas e colunas utilizadas no *trigger*, e o segundo grupo, formado pela tabela *TRIGGERS*, armazena

informações sobre o evento, condição e ação do *trigger*. A limitação do repositório ocorre em função do modelo de regras adotado. O modelo de regras da linguagem SQL3 não permite a definição de regras com duas ou mais ações, ou a especificação de eventos compostos (WIDOM, 1996; PAVON; VIANA; CAMPOS, 2006), portanto, estas diferenças entre modelos não são contempladas na tabela *TRIGGERS*.

Apesar destas estruturas apresentadas pela linguagem SQL3 serem sugestões, os SGBDAs comerciais (ORACLE CORPORATION, 2003; POSTGRESQL, 2005) implementam algumas delas e adicionam outras com a finalidade de melhorar a gerência sobre as regras.

2.5.2 Repositório de um SGBDA

Nesta seção são descritas as estruturas utilizadas por um SGBDA comercial. O critério utilizado na seleção do SGBDA Oracle (ORACLE CORPORATION, 2003) é o fato de que é um dos mais utilizados, além de que sua documentação, completa, está disponível no *site* da empresa (ORACLE CORPORATION, 2003). Além disso, este SGBDA é utilizado por várias empresas, em distintas áreas. A versão analisada é Oracle Database 10g Release 1.

A linguagem de regras deste SGDBA tem por base o modelo de regras sugerido pela linguagem SQL3, porém apresenta algumas variações. Na Figura 4, é apresentado o repositório de regras do Oracle.

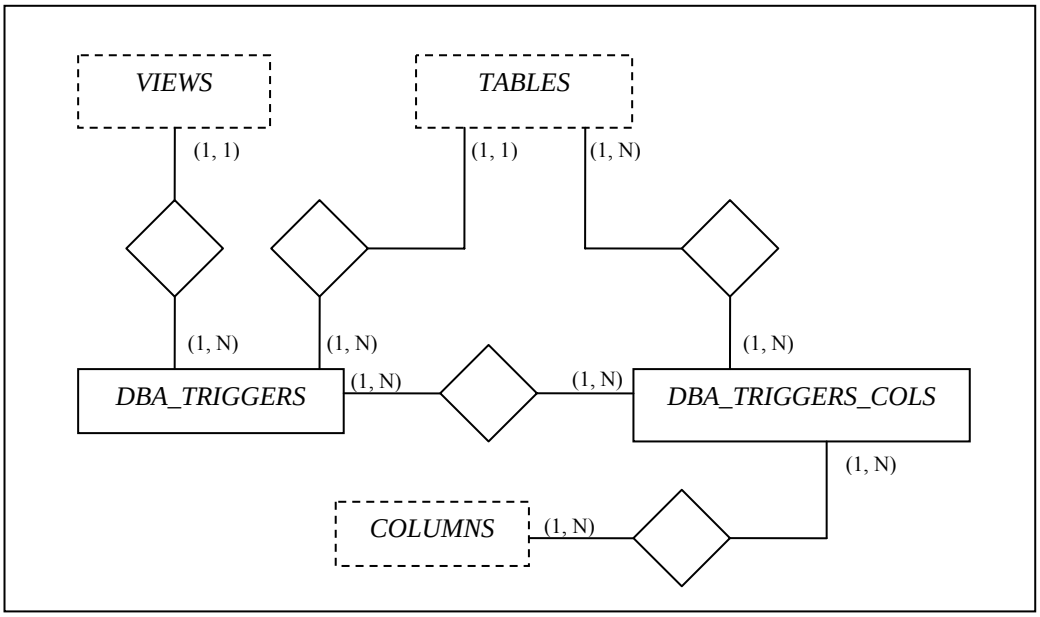

Figura 4 – Repositório de Regras do Oracle

O dicionário de dados do Oracle é composto por duas tabelas principais, DBA_*TRIGGERS* e DBA_*TRIGGERS_COLS*. A tabela DBA_*TRIGGERS* descreve todos os *triggers* do banco de dados e a tabela *DBA_TRIGGERS_COLS* descreve as colunas dos *triggers* do banco de dados. Existem outras tabelas para armazenar informações sobre os *triggers* tais como *USER_TRIGGERS* e *USER_TRIGGERS_COLS*, utilizadas para armazenar as mesmas informações das tabelas apresentadas na Figura 4, com a diferença de que o acesso fica restrito ao usuário corrente. Portanto, não acrescentam quaisquer informações à análise do repositório do Oracle, razão pela qual não são apresentadas na Figura 4.

A primeira tabela, *DBA_TRIGERS*, contém os principais elementos declarados na sintaxe do *trigger*, tais como o seu tempo de ativação, a especificação do evento, da condição e de sua ação, o estado em que se encontra o *trigger* (ativo ou inativo), entre outros. A segunda tabela *DBA_TRIGGERS_COLS* contém informações sobre as colunas utilizadas no *trigger*. No repositório da Figura 4, são apresentados alguns dos meta-dados referenciados pelos *triggers*, representados pelos retângulos pontilhados e na Tabela 2.8 são apresentados os principais elementos da tabela *DBA_TRIGGERS*.

Tabela 2.7 - Conteúdo da tabela *DBA_TRIGGERS*

Coluna	Descrição da coluna
Owner	Proprietário do *trigger*
Trigger_name	Nome do *trigger*
Trigger_type	Especificação do tempo de ativação (*before* ou *after*) e da granularidade de processamento (*each row* ou *statement*). Ex. *before statement, after statement, before each row*, entre outros
Triggering_event	Operação definida como evento (DDL, DML ou eventos de BD).
Base_objetct_type	Tipo de objeto sobre o qual é definido o evento (ex. *table*, *view*, etc.)
Table_name	Nome da tabela ou view sobre o qual é definido o evento
Referencing_names	Nomes especificados para referenciar os valores de transição *new* e *old*
When_clause	Condição do *trigger*
Status	Indica o estado o *trigger* (habilitado ou desabilitado)
Action_type	Indica se a ação é um conjunto de sentenças SQL (PL/SQL) ou uma chamada a um procedimento armazenado (CALL).
Trigger_body	Ação do *trigger*

A tabela *DBA_TRIGGERS* contém um registro para cada definição de *trigger*. Cada um destes registros tem associado o proprietário do *trigger*, o nome e o tipo do *trigger*,

formado pelo tempo de ativação e sua granularidade de processamento. O evento pode ser definido por meio das operações de manipulação de dados (*insert*, *update*, *delete*), operações de definição de dados (*create*, *alter*, *drop*) ou operações de banco de dados (*logon*, *logoff*, *startup*, *shutdown*, *servererror*). O SGBDA Oracle permite especificar eventos compostos, os quais representam uma combinação de eventos simples, por meio do operador *booleano OR.* Não é permitido conjugar eventos de natureza distinta. Por exemplo, eventos que correspondem a operações de manipulação de dados podem ser combinados entre si com o operador *OR,* o mesmo ocorre com operações de definição de dados ou operações de banco de dados.

As informações armazenadas no atributo *trigger_type* estão associadas às informações do atributo *triggering_event*, por exemplo, um *trigger* pode ser disparado mais de uma vez (*for each row*) quando as operações definidas no evento são operações de manipulação de dados. O atributo *status* armazena informações sobre o estado do *trigger*, e por meio de operações de gerência, um usuário pode habilitar ou desabilitá-lo. A condição e a ação são armazenadas integralmente nos atributos, *when_clause* e *trigger_body*, respectivamente, limitado apenas pela capacidade do tipo de dado especificado para o atributo; a condição pode ter no máximo 4kbytes e a ação um tamanho máximo de 32kbytes. A tabela 2.7 apresenta as informações sobre as colunas das tabelas utilizadas no *trigger*.

Tabela 2.8 - Conteúdo da tabela *DBA_TRIGGERS_COLS*

Coluna	Descrição da coluna
Trigger_name	Nome do *trigger*
Table_name	Nome da tabela que contém a coluna referenciada no *trigger*
Column _name	Nome da coluna referenciada no *trigger*
Column_list	Indica se é utilizado ou não no evento *update* (Y ou N)

Nesta tabela é inserido um registro para cada atributo especificado no *trigger*. Quando se trata de um evento definido com *update*, as colunas deste evento são identificadas por meio do atributo *column_list*. A complexidade do repositório é função do modelo de linguagem definido para a especificação do *trigger*. Um exemplo de informação, que não contempla o dicionário de dados do Oracle, é relacionamento entre regras. Ela é importante para identificar quais regras podem ser disparadas por outras regras, sem a necessidade de realizar uma análise aprofundada sobre as tabelas do dicionário de dados.

2.6 GERÊNCIA DE REGRAS

As linguagens de regras dos SGBDAs fornecem um conjunto de operações sobre as regras ou partes destas, de tal forma que permitam gerenciar as informações armazenadas sobre as mesmas no repositório de dados (TURKER, 2001). Um repositório de dados contém informações sobre as regras e por meio do SGBDA estas informações estão disponíveis, tanto para os usuários quanto para as aplicações. Desta forma, eles podem recuperar informações sobre as regras da mesma forma que o fazem com os dados. A linguagem SQL3 sugere um conjunto de operações de gerência, que em sua maioria são implementadas pelos SGBDAs comerciais. Nas seções seguintes são apresentadas as operações de gerência proposta pela linguagem SQL3, e pelos SGBDAs Oracle e Starburst (WIDOM; 1996). A escolha destes dois SGBDAs é pelo fato de que implementam o modelo relacional e suas linguagens de regra (sua sintaxe e seu modelo de execução) serem uma versão estendida das operações de gerência da linguagem SQL3.

2.6.1 Operações de Gerência Proposta pela Linguagem SQL3

A Linguagem SQL3 (TURKER, 2001) sugere um conjunto de operações de gerência para a definição e exclusão de uma regra, do dicionário de dados:

- Create *Trigger* – Este comando cria uma regra no dicionário de dados.
- Drop *Trigger* – este comando exclui uma regra do dicionário de dados

Após a definição de um *trigger*, o usuário pode, somente, eliminá-lo, não existe a possibilidade de alterar o *trigger* ou parte do mesmo. É possível consultar dados referentes a um *trigger*, como por exemplo o seu evento ou o corpo do *trigger*, representado pela sua condição e ação.

2.6.2 Operações de Gerência dos SGBDAs

As operações de gerência de regras oferecidas pelos SGBDAs têm como referência as operações sugeridas pela linguagem SQL3. No SGBDA Oracle (ORACLE CORPORATION, 2003), as operações para a gerência de regras são definidas como operações para criar, excluir, habilitar e desabilitar uma regra. Na Tabela 2.9 são apresentadas as sintaxes destas operações.

Tabela 2.9 - Operações de Gerência no SGBDA Oracle

Nome da operação	Sintaxe da operação
Exclusão de uma regra	*DROP TRIGGER* <nome do *trigger*>;
Habilitar uma regra	*ALTER TRIGGER* <nome do *trigger*> *ENABLE*;
Habilitar todos os *triggers* associados a uma tabela	*ALTER TABLE* <nome tabela> *ENABLE ALL TRIGGERS*
Desabilitar uma regra	*ALTER TRIGGER* <nome do *trigger*> *DISABLE*;

Estas operações variam de um SGBDA para outro. No Starburst[4] (WIDOM, 1996; PATON, 1998), a linguagem para definição e manipulação de regras consiste de cinco comandos, a saber: *create rule, alter rule, deactivate rule, activate rule* e *drop rule*. Considerando que neste SGBDA é possível a criação e manipulação de um conjunto de regras, então outras operações são definidas para dar suporte a estes grupos, como *create ruleset*, *alter ruleset* e *drop ruleset*. O comando *alter rule* é usado para mudar os componentes da regra após ela ter sido definida.

A sintaxe para criação de uma regra e a sintaxe para a sua correspondente alteração são apresentadas na Tabela 2.10.

Tabela 2.10. Comando para criar e alterar regras em Starburst

Sintaxe para criação	Sintaxe para alteração
Create rule <nome da regra> *on* <nome da tabela>	*Alter rule* <nome da regra> *on* <nome da tabela>
When operação definida como evento	[*if* <condição>]
[*if* <condição>]	[*then* <ação>]
then <ação>	[*precedes* <lista de regras>]
[*precedes* <lista de regras>]	[follows < lista de regras >*]*
[*follows* < lista de regras >*]*	[nopriority < lista de regras >]

O comando *create rule* é usado para definir uma nova regra. Cada regra é definida sobre uma tabela. Os colchetes indicam cláusulas que são opcionais. A cláusula *when* especifica um ou mais eventos, que podem ser operações de manipulações de dados (*insert, update* e *delete*). As cláusulas *precedes* e *follows* são usadas para especificar ordem de prioridade entre regras. O comando *alter rule* segue a mesma sintaxe para a criação de regra, na qual a cláusula *nopriority* é usada para remover a prioridade da regra. Quando a cláusula

[4] Protótipo extensível relacional SGBDA desenvolvido pela IBM

when necessita ser alterada, na sintaxe de criação da regra, então é necessário excluí-la e criá-la novamente. Para excluir uma regra, é usado o seguinte comando:

Drop rule <nome da regra> *on* <nome da tabela>

As regras podem ser ativadas e desativadas mediante os seguintes comandos:

Deactivate rule <nome da regra> *on* <nome da tabela>

activate rule <nome da regra> *on* <nome da tabela>

Os comandos para criação, adição e exclusão de um conjunto de regras são especificados pela sintaxe a seguir:

create ruleset <nome do conjunto de regras>

alter ruleset <nome do conjunto de regras>

[*addrules* <lista de regras>]

[*delrules* <lista de regras>]

Cada regra pode estar em qualquer grupo de regras e cada conjunto de regras pode conter qualquer número de regras. Para excluir um conjunto de regras, é necessário utilizar o seguinte comando:

Drop ruleset <nome do conjunto de regras>

O Starburst, além das operações apresentadas, dispõe de um comando para o processamento de regras. Este comando pode ser usado pelo usuário para processar uma regra ou um conjunto delas. A sintaxe deste comando é apresentada a seguir (PATON; DIAZ, 1999; ELMASRI; NAVATHE, 2005).

Process rules

Process ruleset <nome do conjunto de regras>

Process rule <nome da regra>

Apesar de que a linguagem de regras proposta pelo Starburst apresenta mais opções para o gerenciamento de regras, este SGBDA não implementa o conceito de granularidade de regras por linha e nem por conjunto (WIDOM, 1996; PATON, 1998), conceito implementado tanto na linguagem SQL3, quanto nos SGBDAs comerciais.

2.7 CONCLUSÕES

Considerando a base apresentada neste capítulo, conclui-se que os dicionários de dados disponíveis para a representação de regras do tipo ECA e EA, utilizadas pela linguagem SQL3 e pelos SGDBAs comerciais, não são suficientes para oferecerem a infra-estrutura necessária para viabilizar outras operações de gerência, tais como operações para alteração do evento, alteração da condição além das operações atualmente disponíveis: criar, excluir, habilitar e desabilitar uma regra. Isto ocorre em função de que as regras são armazenadas em um ambiente utilizado para armazenar dados, com uma infra-estrutura dedicada a estes dados.

Não existe uma concepção clara e direcionada, para regras, que corresponda ao armazenamento de regras em um ambiente propício para elas, da mesma forma como ocorre para os dados. A elaboração de um repositório de regras, vinculado a um modelo de regras, capaz de oferecer um conjunto de operações de gerência, para definir e alterar regras, é fundamental. Este repositório de regras teria as estruturas para o armazenamento das regras, e um conjunto de meta-regras para garantir a consistência das regras armazenadas no repositório e seria manipulado pelas operações de gerência, da mesma forma como os dados o são, no dicionário de dados. O objetivo é oferecer outro ambiente para que o usuário possa definir suas regras de negócio, além do ambiente que comumente ele define. Bajec (2005) apresenta um conjunto de questões, ainda pendentes, relacionadas com a especificação, implementação, armazenamento e gerência de regras de negócio. Esta tese está orientada a estas duas últimas questões, sendo o foco principal a gerência de regras, de tal forma que seja possível minimizar a impedância atualmente existente, por parte dos usuários, em gerenciar as regras de negócio, definidas no banco de dados (BAJEC; KRISPER, 2005).

Existem outros tipos de regras sugeridos na literatura (PAVON, 2005) como um meio para flexibilizar a representação de regras mais complexas, sendo que os repositórios propostos não oferecem a infra-estrutura necessária para o armazenamento destas regras, como também falta o suporte para definir um conjunto de operações de gerência de regras, que atuem sobre estes novos tipos de regras.

No próximo capítulo, é apresentada uma análise detalhada do meta-modelo e do modelo de regras adotados, que foi sugerido como extensão do modelo de regras adotado pela linguagem SQL3. Este modelo é a base para a especificação de um repositório de regras. Sobre este repositório são definidas as operações de gerência.

CAPITULO 3. MODELO DE REGRAS

3.1 INTRODUÇÃO

Neste capítulo, apresenta-se e analisa-se um modelo de definição de regras. Este modelo é a base para a elaboração da proposta de um repositório de regras, apresentado no capítulo seguinte. O modelo de definição de regras é o resultado da elaboração de um conjunto de características que foram identificadas na linguagem SQL3, analisadas e depois estendidas, e apresentadas em um meta-modelo de regras (PAVON, 2005).

Inicialmente é descrito o meta-modelo de regras, com o objetivo de ilustrar as características dos tipos de regras de negócio freqüentemente utilizadas em sistemas de informação. Logo a seguir descreve-se o modelo de definição de regras. Este modelo, em conjunto com o modelo de execução, formam o modelo de regras, elaborado por Pavón (2005). O modelo de definição, ou sintaxe da regra, contribui para a identificação dos componentes que serão armazenados no repositório de regras. Além deste modelo, é descrito o modelo de execução com a finalidade de descrever o comportamento destas regras. Por meio da análise deste comportamento, é possível identificar algumas informações que auxiliam na elaboração das meta-regras, necessárias para garantir a consistência do repositório.

Uma vez que meta-modelo e modelo sejam descritos, são realizadas ponderações sobre quais aspectos são efetivamente considerados na especificação do repositório de regras. O meta-modelo e o modelo de definição de regras são mencionados no texto como “meta-modelo de regras adotado” e “modelo de regras adotado”. As considerações sobre a linguagem SQL3 que não foram consideradas no modelo proposto e podem ser úteis para a elaboração do repositório de regras, serão descritas neste capítulo.

3.2 META-MODELO DE REGRAS

O meta-modelo de regras adotado neste trabalho (PAVON, 2005) é o resultado da análise de um conjunto de meta-modelos sugeridos na literatura (HERBST; MYRACH, 1995; AMGHAR; MEZIANE; FLORY, 2000; TORRICO; TANAKA; MOURA, 2000). Neste meta-modelo são considerados todos os tipos de regras necessários para dar suporte às regras de negócio expressas em linguagem natural, principalmente as regras de derivação, regras

operacionais e regras estímulo-resposta (PAVON; VIANA; CAMPOS, 2006). As **regras conceituais** (regras que são representadas por meio de modelos de dados como diagrama de classe, diagrama entidade-relacionamento, tabelas) e **regras de afirmação** (representadas por meio de restrições de integridade, como *check, domain* e *assertions*) não são representadas neste meta-modelo, visto que um sistema gerenciador de banco de dados ativo possui suporte apropriado para a representação destes tipos de regras.

As **regras de derivação** caracterizam-se por permitir a tomada de decisões, e portanto, são mapeadas para procedimentos, contendo sentenças do tipo *if-then-else*. As **regras operacionais** indicam uma seqüência de operações que devem ser realizadas com um objetivo específico e, por conseguinte, são formadas somente por ações, e as **regras estímulo-resposta** correspondem a regras ECA (evento-condição-ação). No nível de implementação, estas regras ECA na linguagem SQL3 correspondem aos ***triggers***.

As **regras estímulo/resposta** são definidas por meio de um evento implícito (E), uma condição (C) (opcional), uma ação obrigatória (A_1) e uma opcional (A_2). Para as regras que possuem somente uma ação (A), esta é equivalente à ação A_1. A ação opcional é executada quando a condição é avaliada como falsa. Neste grupo, os elementos obrigatórios são o evento (E) e a ação (A) da regra. Estas regras são classificadas como regras **tipo 1** para diferenciar do outro grupo de tipos de regras, **tipo 2**, cujo evento é definido explicitamente pelo usuário, por meio da **operação *Fire***. Esta operação foi criada para tratar o disparo de regras de derivação e operacional, identificadas no modelo de regras adotado

Na Tabela 3.1, são apresentados os tipos de regras e suas correspondentes representações no nível de negócio.

Tabela 3.1 - Tipos de regras em diferentes níveis de abstração

Tipos de regras no nível de domínio do negócio	**Tipos de regras no nível de especificação**	**Classificação em nível de tipos de regra**
Estímulo/Resposta	ECA, EA, ECA_1A_2	Tipo 1 (evento implícito)
Derivação	CA, CA_1A_2	Tipo 2 (evento explícito)
Operacional	A	

Como os elementos das regras de derivação e operacional são condição-ação e ação, respectivamente, houve a necessidade de criar uma operação que fizesse o papel de gatilho, disparando as regras no momento em que fossem solicitadas. Neste conjunto de regras o único

elemento obrigatório é a ação (A).

As regras do tipo 1, possuem **eventos de banco de dados** e os **eventos temporais**. Os eventos de banco de dados são classificados segundo as seguintes operações de banco de dados:

- *select*, *update*, *insert* ou *delete* sobre uma tabela.
- *alter*, *drop*, *rename* de uma tabela.
- *logon* ou *logoff*
- Chamada a um procedimento
- Chamada de uma função

Os eventos temporais podem ser definidos como temporais absolutos ou relativos. Neste meta-modelo assume-se que os eventos sejam temporais absolutos, ou seja, eventos que correspondem a um ponto específico no tempo, da mesma forma como é especificado na linguagem SQL3.

Na Figura 5 apresentam-se os elementos do meta-modelo, para as regras do tipo 1, por meio de um diagrama entidade-relacionamento.

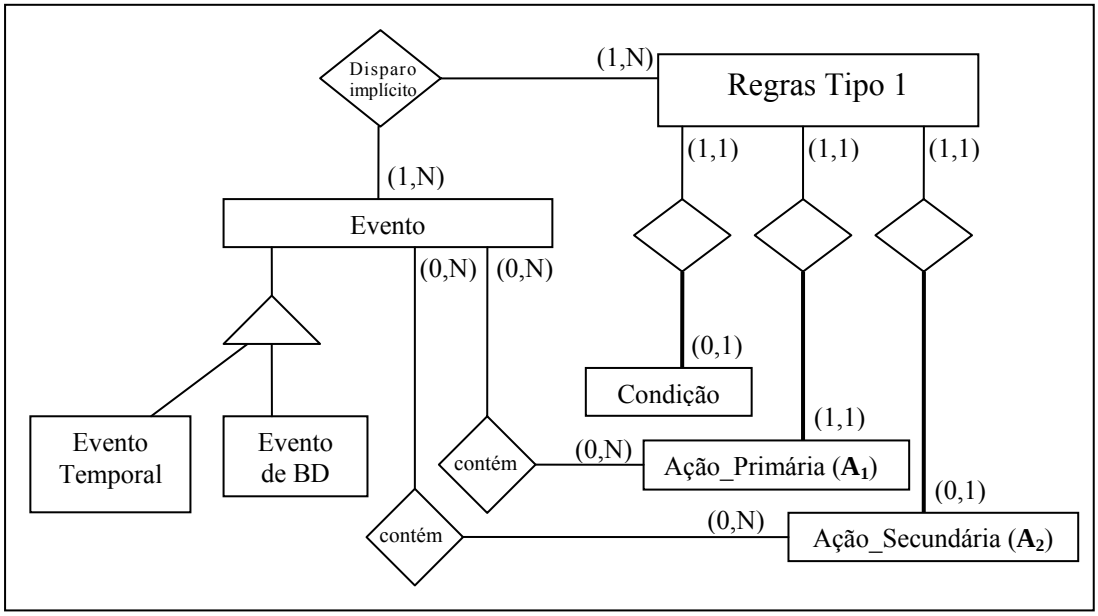

Figura 5 - Representação de regras do tipo 1(adaptado de PAVON, 2005)

Uma característica deste tipo 1 de regra que o diferencia das regras do tipo 2 é o fato de que um evento pode disparar uma ou mais regras e que uma regra pode conter mais de um evento de manipulação de dados. Uma regra do tipo 1 não pode disparar a si mesma, ou seja, considerando que uma regra deste tipo tenha em sua definição um evento de banco de dados e da mesma forma, em sua ação contenha uma operação idêntica à operação de banco de dados,

definida em seu evento, então o auto disparo não é permitido pelo SGBDA.

Na Figura 6 são apresentadas as características de regras do tipo 2. Estas regras são constituídas de uma condição (opcional), uma ação primária (A_1), obrigatória, e uma ação secundária (A_2), opcional. As regras do tipo 2, são executadas por um evento explícito, definido pelo usuário. A ação secundária é especificada para regras do tipo CA_1A_2, sendo que a ação A_2 é executada quando a condição desta regra for avaliada como falsa. As regras deste tipo têm uma relação de um para um com o evento que a dispara, assim, uma regra é disparada por um evento e um evento pode disparar somente uma regra. As ações destas regras podem conter outras regras do mesmo tipo ou regras do tipo 1 e vice-versa. Além disso, é permitido o uso de procedimentos e funções.

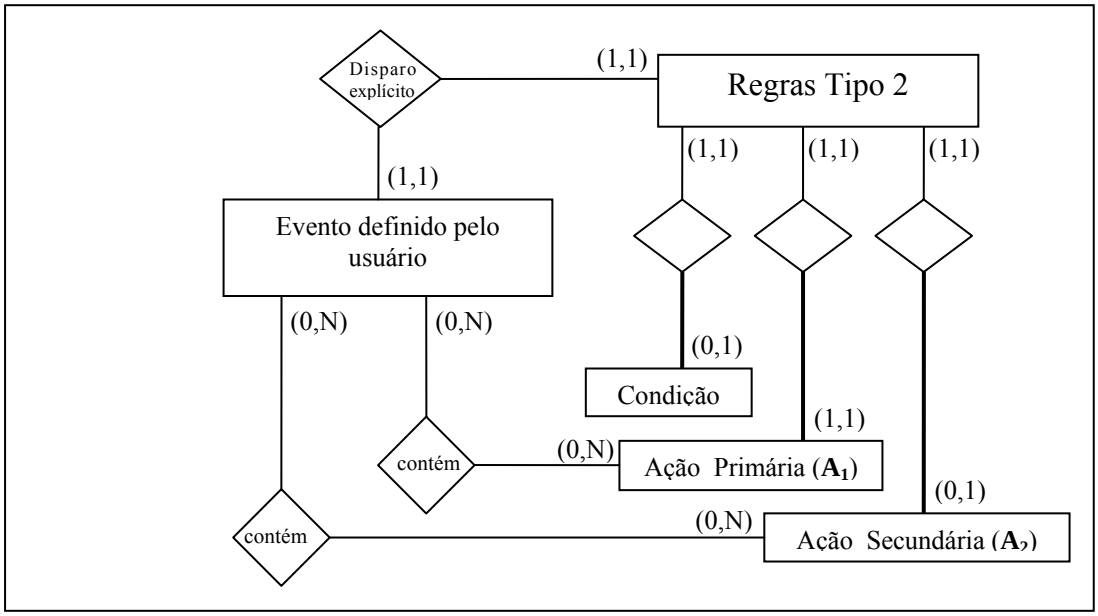

Figura 6 - Representação de regras do tipo 2 (adaptado de PAVON, 2005).

Na Figura 7 é ilustrado o meta-modelo de regras contendo os tipos 1 e 2 de regras, unificados em um único meta-modelo. Nele, a entidade "Entidade" representa o conjunto de tabelas do banco de dados sobre as quais a regra atua, e a entidade "Regra" representa os tipos 1 e 2, de regras.

As regras que estão diretamente relacionadas com as entidades do banco de dados, são representadas pelo relacionamento "referencia". Desta forma, cada regra pode referenciar uma ou mais tabelas do banco de dados, seja no evento ou na ação da regra, como também existem regras que não têm vínculo com estas entidades. O fato de que exista regra sem vínculo com objetos do banco de dados, torna a regra **independente dos dados**, o que por sua vez, eleva as

regras ao mesmo nível dos dados, no banco de dados.

Toda regra está associada, obrigatoriamente, à definição de um evento, seja ele, um evento de banco de dados, um evento temporal ou um evento definido pelo usuário. Porém, somente os eventos de banco de dados podem disparar mais de uma regra. Com relação aos eventos temporais, estes são especificados somente na cláusula "evento", das regras do tipo 1.

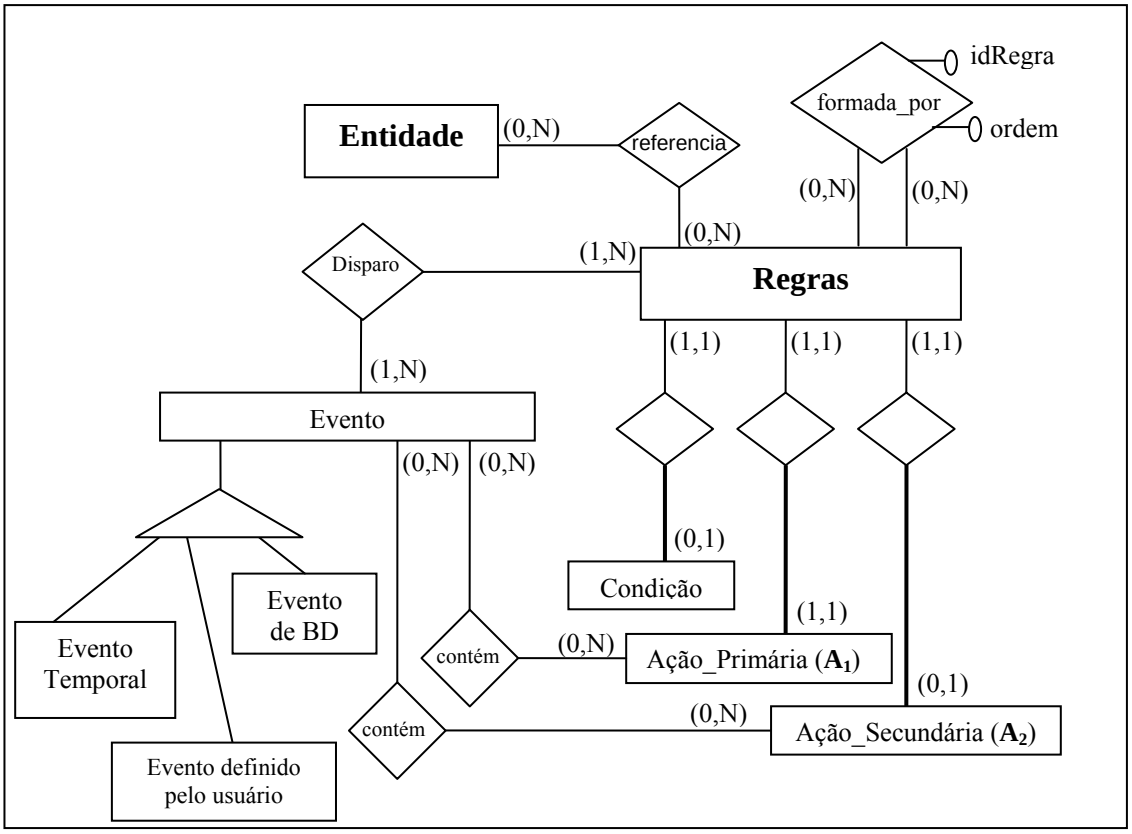

Figura 7 - Meta-modelo de regras (PAVON, 2005)

Regras podem conter, em sua ação, operações de banco de dados que podem desencadear o disparo de outras regras, promovendo um encadeamento de regras. Um encadeamento de regras implica que uma regra pode disparar outras regras automaticamente, sem a intervenção do usuário. Para isto, basta que um evento, sob a forma de uma operação de banco de dados ou a operação *Fire* seja detectado pelo SGBDA. O relacionamento "contém" indica a possibilidade de existência destas operações de banco de dados ou *Fire*, na ação, primária ou secundária, da regra. Quando um destes eventos dispara um conjunto de regras, simultaneamente, é necessária uma política para identificar qual destas regras deve ser disparada em primeiro lugar. Doravante, este conjunto de regras será chamado de conjunto de regras simultâneas. No final desta seção é apresentado um exemplo sobre este conjunto de regras simultâneas.

Uma regra pode ser formada por outras regras e este relacionamento é representado pela associação “formada_por”, cujos atributos são: idRegra e ordem. Estes atributos são utilizados para armazenar informações sobre a ordem de precedência das regras, conforme é exemplificado na regra R1 da Figura 8, na qual a regra R1 é do tipo estímulo-resposta e as demais regras, R2, R3 e R4 são do tipo operacional ou derivação..

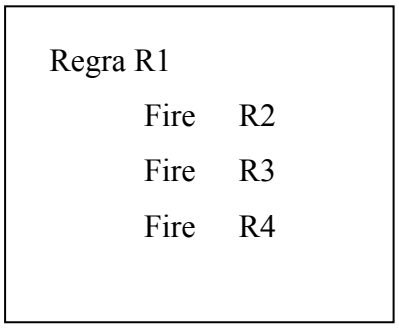

Figura 8 – Regra R1 composta de três regras: R2, R3 e R4

As regras R2, R3 e R4 são disparadas por meio da operação *Fire*. No meta-modelo, esta operação é representada pelo “evento definido pelo usuário”.

Considerando que a regra R1 foi disparada por meio de um evento de banco de dados, então, a seqüência de execução das regras internas à regra R1, ocorre de cima para baixo, primeiro a regra R2, logo a seguir R3, e depois a regra R4. Esta ordem define a seqüência de execução das regras, independente do momento no tempo (*timestamp)* em que estas regras foram criadas. Esta característica é uma contribuição do modelo adotado e que foi estendida, em relação ao tratamento de regras que são disparadas ao mesmo tempo, da linguagem SQL3.

Na linguagem SQL3, a definição de data e hora de criação da regra dita a ordem de execução das regras simultâneas. Caracterizar o disparo de uma regra em função da data e hora torna o processo de manutenção destas regras difícil. Esta dificuldade está relacionada com dois aspectos. O primeiro aspecto, é o fato de que não existem operações de gerência que auxiliem na manutenção das regras. As operações oferecidas são utilizadas para criar ou eliminar uma regra. O segundo aspecto, está relacionado com a data de criação da regra, pois, toda vez que uma regra é criada no dicionário de dados, ela recebe uma nova data e hora, geradas pelo sistema. Como não existem operações para alteração de uma regra ou parte dela, é necessário recriar a regra, e desta forma, ela deve ser novamente e integralmente processada.

Por exemplo, considerando o contexto da regra R6, ilustrado na Figura 9, e que esta possui uma operação de banco de dados definida em sua ação, e supondo que o negócio exija

que as regras R7, R8, e R9 sejam disparadas em função desta operação de banco de dados, então, as regras serão disparadas simultaneamente. Portanto, a regra que contiver a data mais antiga será dispara em primeiro lugar, e assim sucessivamente. Levando em conta que a regra R8 seja modificada, isto é, criada novamente, e novamente compilada, como parte do requisito de negócio, então, esta regra, recebe um novo valor de tempo de criação, fazendo com que a ordem de execução das regras varie (R8, R7 e R9), levando a uma seqüência de execução diferente em comparação com a seqüência anterior (R7, R8 e R9). A ordem de execução assume que a data/hora mais antiga é executada em primeiro lugar.

Esta nova seqüência deixa de corresponder ao contexto do negócio, para o qual foi especificado. Quando o conjunto de regras tende a ser grande, a manutenção da lógica do negócio torna-se uma tarefa mais complexa de ser mantida. Uma forma de minimizar esta complexidade é empregar a operação *Fire*. O uso desta operação facilita a gerência destas regras, porque elas são executadas, explicitamente, pela operação de disparo, desvinculando o *timestamp* de sua execução e, como conseqüência, facilitando a manutenção destas regras. Ao mesmo tempo é possível especificar contextos de negócios diferentes, para um mesmo conjunto de regras. Por exemplo, seja o contexto da regra R5 a seguir:

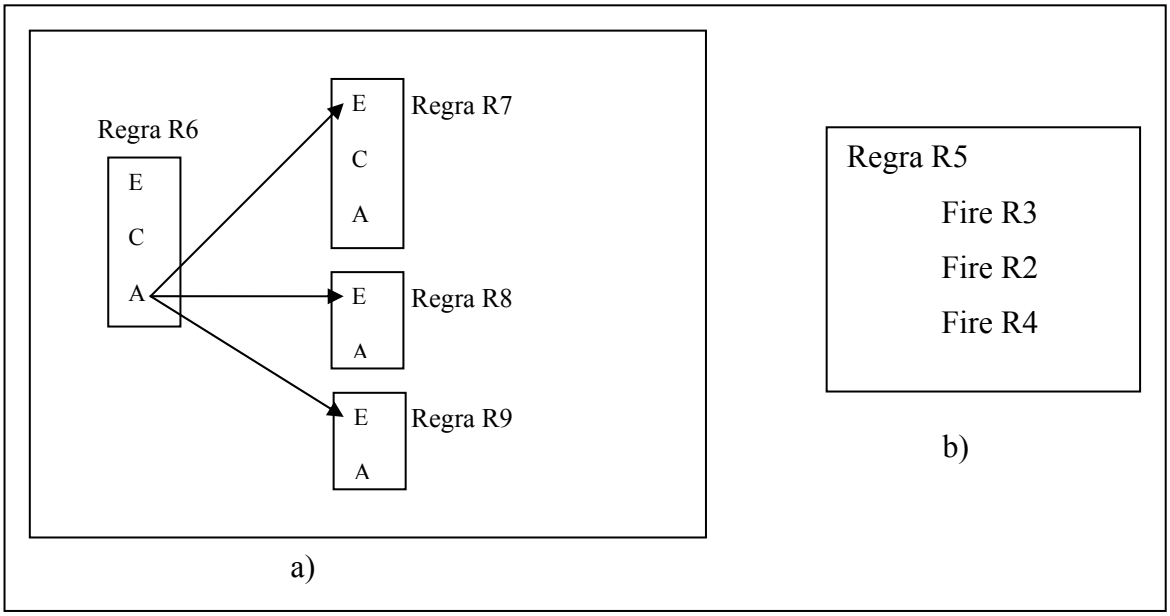

Figura 9 – Dois tipos de representação de disparos de regras. Na alínea a, é ilustrado um exemplo implícito no qual a regra R6 dispara a regra R7, R8 e R9, e estas regras estão sujeitas a uma mudança na ordem de execução. Na alínea b eventos explícitos, sendo que a regra R5 compõe-se de três regras: R3, R2 e R4

A regra R5 contém as mesmas regras especificadas na regra R1, da Figura 8, porém a ordem de execução destas regras é diferente, apresentando contextos de negócio diferentes. A

diferença da regra R5 para a regra R6 está em que a regra R5 **contém um conjunto de regras** (R3, R2 e R4) e a R6 contém um evento que **dispara um conjunto de regras** (regras simultâneas) nos quais seus eventos são iguais ao evento definido na ação de R6.

3.3 MODELO DE REGRAS ADOTADO

Um modelo de regras compreende o modelo de definição e o modelo de execução de regras. O modelo de definição é representado pela sintaxe da regra, e define os elementos que são utilizados para a especificação das regras, do tipo 1 e do tipo 2, e o modelo de execução determina as características comportamentais destes tipos de regras.

O modelo de regras utilizado neste trabalho é uma proposta de extensão e adaptação do modelo de regras sugerido pela linguagem SQL3 para a definição e execução de regras em um sistema de banco de dados ativos. Este modelo de regras apresenta dois tipos de operações de gerência de regras: Uma operação para a definição da regra e uma operação para disparar explicitamente uma regra do tipo 2. A operação para eliminação de regras sugerida pelo modelo adotado é a mesma apresentada pela linguagem SQL3.

3.3.1 Modelo de Definição de Regras

A sintaxe para a representação da definição de regras do modelo adotado emprega os conceitos sugeridos pela linguagem SQL3 e adequa novos conceitos aos já existentes, de tal forma que seu poder de expressividade melhora, em função dos novos elementos que são adicionados à sua estrutura, e que são representados pelas regras do tipo 1 e do tipo 2. Neste sentido, a expressividade da linguagem SQL3 para a definição de regras é um subconjunto dos tipos de regras oferecidos pelas regras do tipo 1, visto que a SQL3 sugere os tipos de regras ECA e EA e os tipos 1, de regras, são representados por regras ECA, EA e ECA_1A_2.

Na Tabela 3.2 é apresentada a sintaxe da linguagem SQL3 e de sua extensão, definidas segundo a notação da meta-linguagem BNF. O que está em negrito representa a extensão à linguagem de regras SQL3.

Tabela 3.2 - Linguagem de regras SQL3 e sua Extensão (VIANA; PAVON; ALMEIDA, 2006)

SQL3	SQL3 Estendida
CREATE RULE <nome-trigger> {before|after} <evento> [REFERENCING <variáveis de transição>] [for each {row|statement}] [when (<condição>)] <ação>	**CREATE RULE** <nome da regra> **[**<tempo de ativação> <evento>**]** [REFERENCING <variáveis-transição>] [<granularidade>] [WHEN **<condição>**] DO **<ação primária>** **[ELSEDO <ação secundária>]**

A linguagem de regras, quando processada pelo SGBDA, resulta em um conjunto de objetos que são armazenados no dicionário de regras. Estes objetos são os elementos das regras e podem ser manipulados por meio de operações de gerência de regras e, como conseqüência, podem ser reutilizados. Esta reutilização está diretamente relacionada com o repositório de regras utilizado para armazenar os elementos das regras (objetos).

No capítulo 4, é apresentado um repositório de regras para armazenar os tipos de regras sugeridos na extensão proposta e os respectivos elementos destas regras.

A seguir, é apresentada a linguagem de regras utilizada neste trabalho, seguido de uma explicação de seus elementos.

A sintaxe para definir uma regra é a seguinte:

<Definição-da-Regra> ::= **CREATE RULE** <nome da regra>
[<tempo de ativação> <evento>**]**
[REFERENCING <variáveis-transição>]
[<granularidade>]
[WHEN **<condição>**]
DO **<ação primária>**
[ELSEDO <ação secundária>]

<tempo de ativação> ::= **ON** | BEFORE | AFTER

<evento> ::= **<evento-BD>** |
<evento-temporal>

<evento-BD> ::= **<evento-dados> |**
<evento-meta-dados> |
<evento-sistema> |
<evento-procedimento>

<evento-dados> ::= <operação-dados> [**{OR <operação-dados>}...**]
ON <nome da tabela>

<operação-dados> ::= **SELECT [OF <lista de colunas>}]**
| INSERT
| DELETE
| UPDATE [OF <lista de colunas>]

<lista de colunas> ::= <nome da coluna>
[{<vírgula> <nome da coluna>}...]

<evento-meta-dados> ::= **<operação-meta-dados>**
[{OR <operação-meta-dados>}...]
TABLE <nome da tabela>

<operação-meta-dados> ::= **ALTER |**
DROP |
RENAME

<evento-sistema> ::= **LOGON | LOGOFF**
[{OR LOGON | LOGOFF}]
OF <nome do usuário>

<evento-procedimento> :: = PROCEDURE <nome do procedimento> |

FUNCTION <nome da função>
[{OR PROCEDURE
<nome do procedimento> |
FUNCTION <nome da função>}...]

<evento-temporal> **::= (<hora>)<data>**

<hora> ::= hh:mm:ss

<data> ::= dd/mm/yyyy | */*/*

<variáveis de transição> ::=
OLD [ROW] [AS] <nome alternativo>
| NEW [ROW] [AS] <nome alternativo>
| OLD TABLE [AS] <nome alternativo >
| NEW TABLE [AS] <nome alternativo >
| CURRENT [ROW] [AS] <nome alternativo>

<granularidade> ::= FOR EACH {ROW | STATEMENT}

<condição> ::= **[<operador unário>]** <elemento-condição>
[{<conector> <elemento-condição>}]**...**

<operador unário> ::= NOT

<conector> ::= AND | OR
<elemento-condição> ::= <condição-SQL3> |
<predicado-variável-de-transição>|

<ação primária> ::= <sentença procedimental SQL> |
CALL <nome do procedimento> |
<bloco SQL> |
<operação-regra> <nome da regra>

<ação secundária> ::= <sentença procedimental SQL> |
CALL <nome do procedimento> |
<bloco SQL> |
<operação-regra> <nome da regra>

<bloco SQL> ::= BEGIN
{<sentença procedimental SQL>
<ponto e vírgula> }...
END;

A linguagem de regras adotada possui quatro componentes, a saber: um evento, uma condição, duas ações (primária e secundária). O único elemento obrigatório é a ação primária, os demais são opcionais.

a) Criação de Regras

Toda regra armazenada no dicionário de dados é única. Seu nome é definido por meio da expressão:

CREATE RULE <nome da regra>.

b) Eventos

A regra, uma vez que é definida e armazenada no banco de dados, é disparada em função da ocorrência de seu evento. Este evento tem associado um tempo de ativação que por sua vez é atribuído apenas às regras do tipo 1, visto que, elas exigem a especificação deste evento em sua definição.

Na linguagem SQL3, este tempo de ativação é definido pelas palavras-chave *BEFORE* e *AFTER*, e são utilizadas conforme o tipo de evento definido em sua sintaxe. O uso de *BEFORE* ocorre quando é desejado que a regra seja disparada antes da execução da operação definida como evento. O uso de *AFTER* significa que a regra foi disparada depois da execução da operação definida como evento. Na linguagem proposta, foi adicionada uma outra palavra-chave ***ON***, em função da adição de eventos de sistemas (evento-sistema). Estes eventos fazem parte dos eventos de banco de dados (evento-BD) e são:

<evento> ::= <**evento-BD**> | <**evento-temporal**>

Os eventos de banco de dados referem-se às operações de banco de dados que podem ser executadas por um sistema de banco de dados ativos. A proposta inclui quatro grupos de eventos de banco de dados:

<evento-BD> ::= <**evento-dados**> |
<**evento-meta-dados**> |
<**evento-procedimento**>
<**evento-sistema**> |

Os **<evento-dados>** são compostos pelas operações de banco de dados *insert*, *update*, *delete* e *select*. Esta última operação foi adicionada em função da necessidade de que aplicações de software necessitam monitorar o acesso às informações armazenadas no banco de dados. Desta forma ela é elevada ao mesmo nível de importância das demais operações utilizadas para a manipulação das informações do banco de dados.

Os **<evento-meta-dados>** são compostos por operações definidas na linguagem de definição de dados da SQL3, utilizadas para definir os meta-dados. As operações que são possíveis de serem especificadas na definição de uma regra são: *alter, drop* e *rename*. Estas operações devem ser utilizadas em conjunto com a definição de uma tabela de banco de dados.

Os **<evento-procedimento>** são compostos por procedimentos ou funções.

Os eventos **<evento-dados>, <evento-meta-dados>** e **<evento-procedimento>** podem utilizar o tempo de ativação *BEFORE* ou *AFTER*.

Por exemplo:

CREATE RULE Pedido_Emprestimo
AFTER UPDATE OR INSERT ON PEDIDO
FOR EACH ROW
DO …

No exemplo anterior, as operações de banco de dados UPDATE OR INSERT indicam que a regra contém dois eventos especificados pelo conector *OR*. A regra que contém esta definição pode ser disparada por um destes dois eventos. O tempo de ativação AFTER implica que a regra é disparada depois que o evento da regra for instanciado (ocorrer). As regras com os eventos **<evento-dados>** podem combinar as operações *insert*, *update* ou *delete*, por meio do conector *OR*, formando um evento composto. Para os eventos **<evento-meta-dados>**, é possível combinar os eventos *alter*, *drop* ou *rename* e por último, os eventos **<evento-procedimento>** podem combinar os eventos relacionados a procedimentos e funções. Não é permitido combinar eventos de grupos diferentes, ou seja, um **<evento-**

dados> com **<evento-meta-dados>** ou evento **<evento-procedimento>.**

Os **<evento-sistema>** são compostos pelas operações *logon* ou *logoff*. A operação *logon* é utilizada para conectar um usuário ao banco de dados e a operação *logoff* é a operação inversa, para desconectá-lo do banco de dados. Estas operações são utilizadas com o tempo de ativação ***ON***, e podem ser combinadas por meio do operador lógico *OR*, conforme o seguinte exemplo:

```
CREATE RULE Conexao_BD
ON LOGON OR LOGOFF OF usuario
DO …
```

O <**evento-temporal**> representa uma instância na linha do tempo, definida por uma hora e/ou data. O tempo definido para este evento é absoluto. A granularidade da hora é definida em termos de horas, minutos e segundos. A granularidade da data é definida em dia, mês e ano. Portanto o tempo absoluto é constituído de dia, mês, ano, hora, minuto e segundo, conforme apresentado a seguir:

<evento-temporal> ::= (<hora>)<data>

<hora> ::= hh:mm:ss

<data> ::= dd/mm/yyyy | */*/*

Na sintaxe da especificação de evento temporal, o '*' (asterisco) representa qualquer hora dentro da data estipulada pelo usuário. O tempo de ativação de uma regra com evento temporal é definido por meio da cláusula *ON*, conforme o seguinte exemplo:

```
CREATE RULE Auditoria_BD
ON (10:00:00) 10/10/2004 ON Empregado
DO
    BEGIN
    CALL PROCEDURE Verificar_Status_Funcionario( )
END;
```

A regra "Auditoria_BD" foi especificada para disparar às 10 horas da manhã do dia 10 de outubro de 2004. A presença da data não é obrigatória e sua ausência indica que a regra é disparada todos os dias. Desta forma é possível definir um evento temporal somente com a

hora em que a regra deve ser disparada.

c) Variáveis de Transição

A especificação de um evento pode incluir também o conceito de variáveis de transição e granularidade da regra. As variáveis de transição são utilizadas para se ter acesso aos valores de cada tupla que são afetados pelas operações de manipulação de dados. As variáveis de transição utilizadas pela linguagem SQL3 são *old* e *new*. A variável *old* é utilizada para ter acesso ao valor prévio do atributo antes de sua atualização no registro. A variável *new* é utilizada para o acesso ao valor futuro do atributo antes de sua atualização no registro. A extensão proposta sugere uma nova variável de transição, definida como *current*. Esta variável de transição é utilizada para se ter acesso ao valor atual do atributo. A variável *current* é utilizada em conjunto com a operação de seleção, e as demais variáveis de transição (*new*, *old*) são utilizadas com as operações de atualização, exclusão e inserção. O exemplo a seguir utiliza a variável de transição *current*.

```
CREATE RULE Acesso_Sal_Chefe
BEFORE SELECT OF salario ON EMP
FOR EACH ROW
WHEN (current.cargo = 'Chefe')
DO .....
```

Neste exemplo, a regra "Acesso_Sal_Chefe" tem acesso aos salários dos empregados, e na condição desta regra é verificado se este salário pertence aos empregados que tenham o cargo de chefe.

A granularidade da regra é especificada para as operações de manipulação de dados. Esta granularidade indica o grau de relacionamento entre a ocorrência do evento e o disparo da regra. Na linguagem de regras proposta, assume-se a mesma granularidade sugerida pela linguagem SQL3.

d) Condições

A condição da regra é especificada pela cláusula **WHEN**, dentro da sintaxe geral de definição de regra. A condição é um componente opcional para os dois padrões de regras. A avaliação da condição, pelo SGBDA, é expressa por um valor *booleano* (verdadeiro ou falso),

independente da quantidade de elementos associados por operadores *AND* ou *OR*. Nesta linguagem estendida, foi sugerido o uso do operador unário *NOT* às informações propostas pela linguagem SQL3. O <elemento-condição> é um predicado na linguagem SQL3, que pode ser uma condição na linguagem SQL3 ou um predicado utilizando variáveis de transição, sendo que neste caso, exige-se que o evento seja um <evento-dados>.

A seguir, é um apresentado um exemplo de condição envolvendo um predicado com variáveis de transição. Neste exemplo, especificamente na condição da regra (*When*), a variável de transição *new* está associada ao atributo qtd_estoque. A ação é executada se e somente se a nova quantidade em estoque for menor ou igual ao estoque mínimo.

```
CREATE RULE Controle_estoque
AFTER INSERT OR UPDATE OF qtd_estoque ON PRODUTO
FOR EACH ROW
WHEN new.qtd_estoque <= estoque_min
DO ...
```

e) Ações

Na linguagem de regras proposta, existem dois tipos de ações, a saber: uma ação primária, obrigatória e uma ação secundária, condicionada à existência da cláusula *WHEN* da regra, ou seja, uma regra que não possui condição, então possui somente uma ação, definida como primária. Isto é válido para os dois padrões de regras. A ação primária é definida pela cláusula *DO* e a ação secundária pela cláusula *ELSEDO*.

A ação primária é executada toda vez que a condição é avaliada como verdadeira, caso contrário, executa-se a ação secundária. Para os tipos de regras constituídos de uma única ação, ou ação primária, o resultado da avaliação da condição, quando falso, não muda o estado do banco de dados.

Os elementos permitidos na especificação de uma ação são: uma sentença procedimental SQL, uma chamada a procedimentos, um bloco SQL ou uma operação de gerencia de regra.

Seja a seguinte regra:

```
CREATE RULE Regra_Estoque
AFTER UPDATE ON PRODUTO
FOR EACH ROW
DO CALL PROCEDURE Verificar_Quant_min(new.cod_prod, new.quant );
```

A regra acima, do tipo EA (evento-ação) é disparada toda vez que uma operação de banco de dados *update* sobre a tabela PRODUTO ocorrer. Após a atualização do produto no estoque, é disparada a sua ação, cuja finalidade é chamar o procedimento Verificar_Quant_min. Este procedimento recebe como parâmetro o código do produto e sua quantidade. Quando o estoque é menor ou igual ao estoque mínimo, alguma ação deve ser tomada.

No exemplo a seguir, apresenta-se a regra R33 (PAVON, 2005), do tipo CAA_2.

```
CREATE RULE R33   /* Avaliar valor solicitado de empréstimo*/
WHEN ( (SELECT valor_max_empr FROM PEDIDO_EMPRESTIMO
          WHERE codcli = :V_Pedido_Empr.codcli) >=
           :V_Pedido_Empr.valorsolicitado)
DO
 BEGIN
    UPDATE EMPRESTIMO
    SET aprovado = 'S'       /* indica que o empréstimo foi aprovado */
    WHERE codcli = : :V_Pedido_Empr.codcli
    AND codpedido = :V_Pedido_Empr.codpedido;
    FIRE R34;
    CALL PROCEDURE Comunica_Cliente('S');
 END;
ELSEDO
 BEGIN
    UPDATE PEDIDO_EMPRESTIMO
    SET aprovado = 'N'       /* indica que o empréstimo não foi aprovado */
    WHERE codcli =  :V_Pedido_Empr.codcli
    AND codpedido = :V_Pedido_Empr.codpedido;
    CALL PROCEDURE Comunica_Cliente('N');
 END;
```

Esta regra apresenta uma condição, uma ação primária e uma ação secundária. Na condição da regra é avaliado o seguinte predicado: “O valor máximo definido para empréstimo é maior que o valor máximo solicitado pelo cliente”. Caso este predicado seja verdadeiro, então se dispara a ação primária, especificada pela palavra-chave *DO*, caso contrário, é disparada a ação secundária, definida por *ELSEDO*.

Nesta sentença, a ação primária contém uma outra regra, conforme apresentado na Figura 9, definida por “R34”, cuja finalidade é definir o valor de empréstimo a ser dado para o cliente. Este valor é proporcional ao perfil apresentado por ele, um dos atributos deste perfil é o seu salário.

Quando uma ou mais regras são definidas a partir de outra regra (regras embutidas em outras regras), considera-se que as regras embutidas herdam os valores ou variáveis definidas na regra hospedeira. Este procedimento minimiza a repetição de variáveis e facilita a escrita das regras.

3.3.2 Modelo de Execução de Regras

O modelo de execução de regras determina as características comportamentais das regras, ou seja, as características envolvidas durante o processamento de regras. Estas características estão associadas ao seu *modus operandi*, que incluem o seu disparo (instanciação do evento) e sua execução (avaliação da condição execução da ação), o mecanismo adotado para a resolução de conflito de regras, modo de acoplamento e granularidade de processamento.

3.3.2.1 Disparo e Execução de Regras

Os disparos das regras do tipo 1 e do tipo 2 apresentam características diferentes, no primeiro caso, as regras são disparadas conforme a especificação do evento em sua sintaxe, enquanto que no segundo caso, as regras são disparadas por um evento definido pelo usuário, por meio da operação *Fire*. Na Figura 10 são apresentadas as etapas do processamento de regras do tipo 1 e do tipo 2.

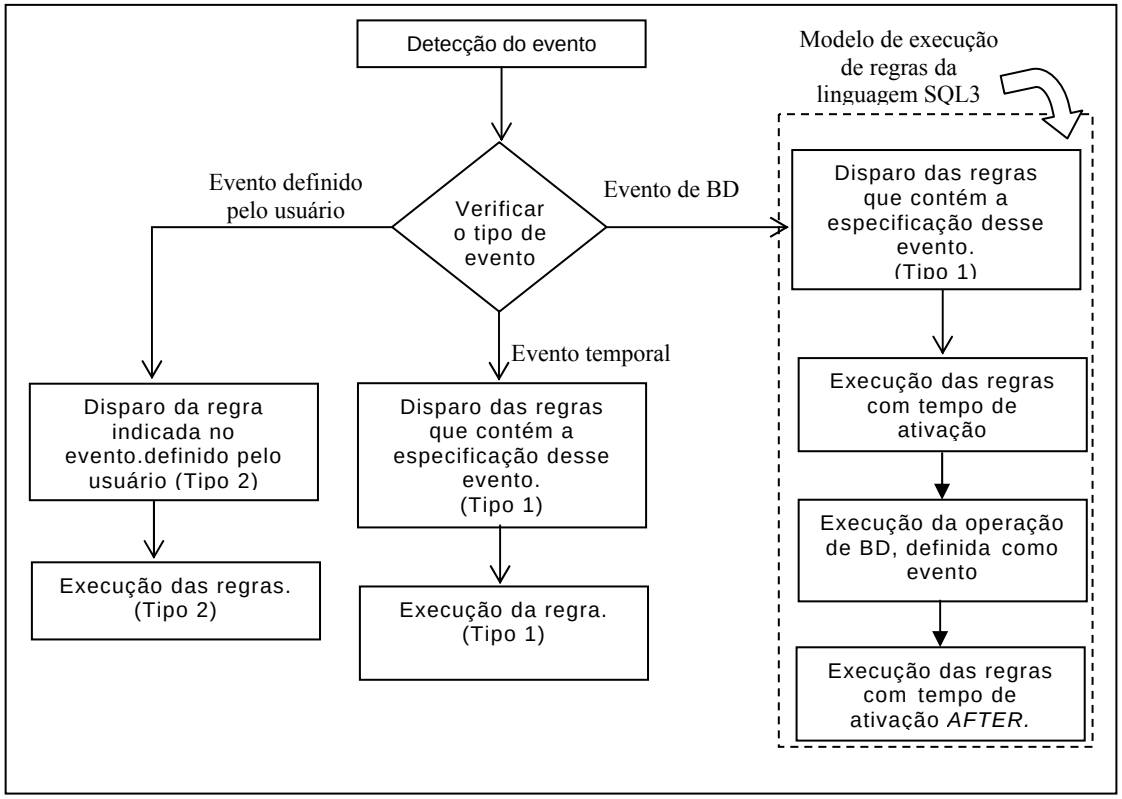

Figura 10 - Etapas do processamento de regras do tipo 1 e tipo 2 (PAVON, 2005).

Uma vez que a regra é disparada, ou seja, o evento é detectado e devidamente identificado, a regra correspondente é executada, da seguinte forma, se o evento é do tipo definido pelo usuário, executa-se uma regra do tipo 2, se o evento é temporal ou evento de banco de dados, executa-se uma regra do tipo 1. O passo seguinte é a avaliação da condição e, de acordo com o resultado desta avaliação, executa-se a ação correspondente. Regras que não possuem a cláusula "condição" executam diretamente sua ação.

Uma vez que as regras do tipo 2 foram disparadas, então elas são imediatamente executadas. Neste tipo, somente uma regra é disparada por vez (ex. *Fire* R3). As regras do tipo 2 apresentam um outro tipo de comportamento, que difere do sugerido pela linguagem SQL3, e que pode ser usado para o tratamento de um conjunto de regras que apresente disparo simultâneo. No item 3.3.2.2 é discutida a resolução de disparos simultâneos de regras, entre outros mecanismos associados ao modelo de execução de regras.

Quando se trata de eventos de banco de dados, as regras que contêm o evento identificado pelo SGBDA são disparadas. Em se tratando do disparo de apenas uma regra, então sua execução ocorre de forma imediata, caso contrário, o SGBDA agrupa as regras em dois conjuntos, as regras com tempo de ativação BEFORE e com tempo de ativação AFTER

(FORTIER, 1999; ISO/IEC 9075-2, 1999; TURKER; GERTZ, 2001);. Logo a seguir, são executadas as regras do primeiro grupo, seguindo o critério de antiguidade destas regras, dado pelo seu tempo de criação, ou seja, data e hora em que foram criadas. Logo a seguir, executam-se as regras do segundo grupo, seguindo o mesmo critério apresentado anteriormente. Esta característica de resolução de conflito é implementada nos SGBDA comerciais, como Oracle e PostgreSQL.

3.3.2.2 Outros Mecanismos Associados ao *Modus Operandi*

Na Figura 11 é ilustrado uma regra definida como R6, do tipo EA, que possui em sua ação uma operação de inserção sobre uma tabela do banco de dados. Esta operação é a mesma operação definida no evento das regras R7, R8 e R9.

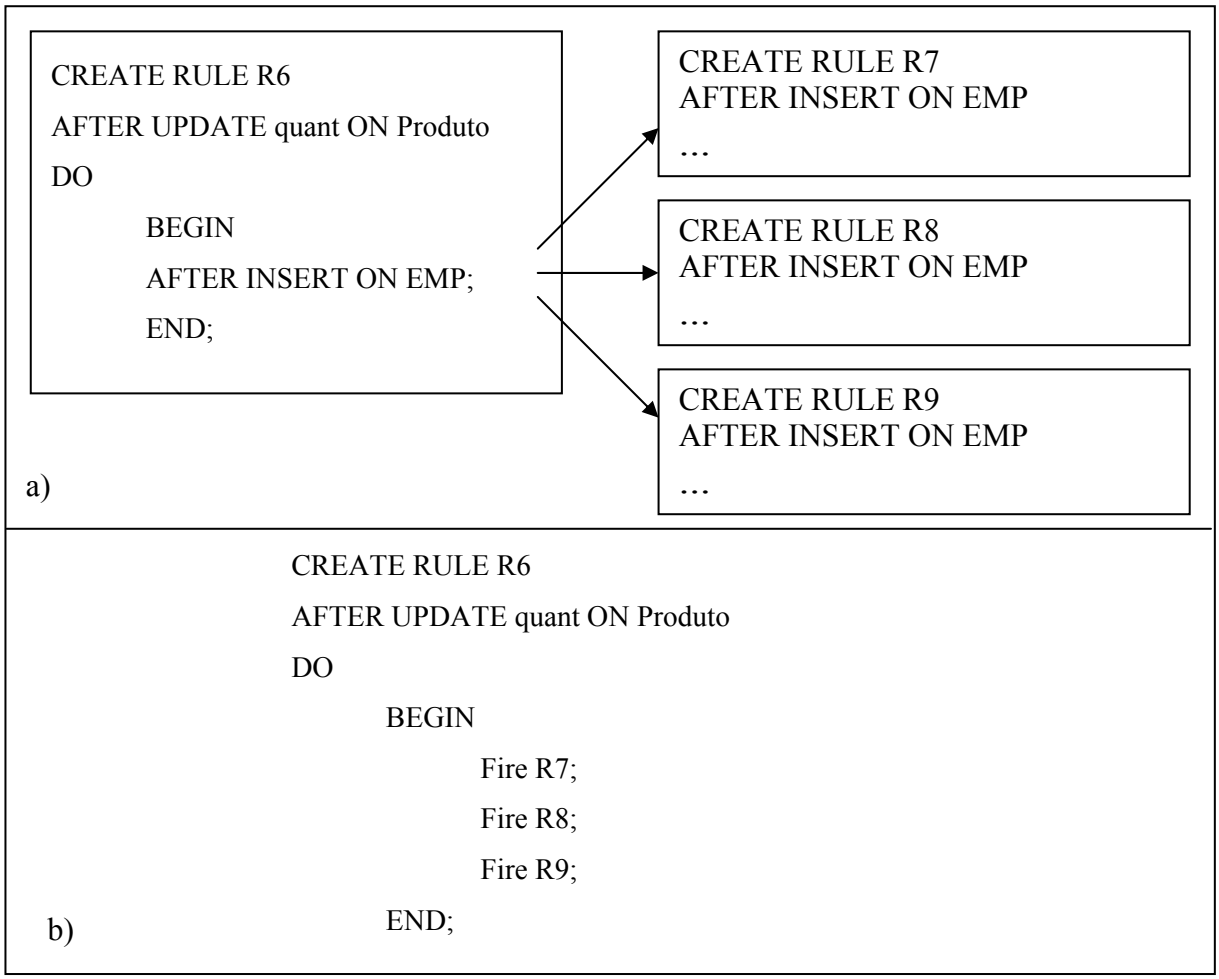

Figura 11 – Regra R1 especificada de duas formas diferentes.

As regras R7, R8 e R9, são especificadas por meio de um evento AFTER INSERT ON

EMP (alínea a). Quando este evento ocorre, todas estas regras são disparadas ao mesmo tempo. A execução destas regras, dentro do conjunto de regras disparadas simultaneamente, depende do critério de execução adotado. Para a linguagem SQL3 a referência é o *timestamp*.

Para solucionar este problema, estas três regras são transformadas em uma única regra R6, na qual a ação desta regra é o conjunto formado pelas três regras. Desta forma, mantém-se a mesma característica do evento e tempo de ativação para todo o conjunto, e define-se uma ordem de execução para elas, de forma explícita, independente do tempo de criação das mesmas. Esta mesma solução pode ser aplicada para eventos temporais.

Além do conflito de regras, existem outros conceitos sobre o processamento de regras, a saber: o acoplamento de regras e a granularidade de regras. Quando uma regra é disparada, e o seu evento instanciado, então, imediatamente a sua condição é avaliada e logo que esta atividade é terminada, inicia-se a execução da ação da regra. A este comportamento, em que as atividades de instanciação do evento e avaliação da condição (quando existir) ocorre de forma imediata, ou até mesmo, entre a avaliação da condição e execução da regra também ocorre de forma imediata, dá-se o nome de acoplamento imediato. O acoplamento de regras pode ocorrer para os pares de componentes da regra (evento-condição e condição-ação). Na linguagem SQL3, também é sugerido este tipo de modo de acoplamento.

A granularidade de processamento das regras pode ocorrer em nível de instância ou em nível de conjunto. Quando é em nível de instância, a condição é verificada para cada registro da tabela afetada e a ação é disparada para cada registro. Quando é em nível de conjunto, a condição é verificada uma única vez e a ação é disparada uma única vez, também. A linguagem SQL3 assume as duas propostas anteriores.

3.4 OPERAÇÃO SOBRE REGRAS

As operações de gerência sugeridas pelo modelo de regras adotado são cinco, a saber: *Fire*, Habilitar, Desabilitar, Excluir e Criar uma regra. Toda regra criada em um banco de dados está em um de dois estados, habilitado ou desabilitado. Uma regra está no estado desabilitado quando está armazenada no dicionário de dados e não pode ser disparada, sob hipótese alguma. O estado habilitado é o inverso do estado desabilitado. Estas operações podem ser definidas tanto na ação primária quanto na ação secundária. Uma regra pode habilitar ou desabilitar outras regras, porém uma regra não pode habilitar ou desabilitar a si mesma.

As sintaxes destas duas operações de gerência são apresentadas a seguir:

ENABLE <nome da regra>;

DISABLE <nome da regra>;

Estas duas operações são mantidas na abordagem adotada, em conjunto com a operação de gerência, definida como ***Fire***. Esta operação tem a função de disparar explicitamente uma regra que é especificada pelo tipo 2. Portanto, esta operação é o evento que dispara as regras CA, CA_1A_2 e A.

A sintaxe desta operação é definida a seguir:

***FIRE* <nome da regra>;**

Esta operação pode ser definida na ação primária ou ação secundária de uma regra. Quando se necessita executar uma regra definida no repositório de dados explicitamente, é necessário que se disponha de um mecanismo que o faça explicitamente, por meio de uma ferramenta que seja construída com este propósito, de auxiliar a execução destas operações de gerência. Quando a regra possui eventos definidos por meio de operações de banco de dados, estão existe uma dependência da ocorrência (implícita) destas operações, para que a regra seja disparada, ou por meio de uma ferramenta, provida pelo próprio SGBDA, executar alguma consulta de banco de dados que gere o disparo da regra.

O modelo de definição de regras adotado, promove novas variações de tipos de regras, porém faltam definir operações de gerência que auxiliem a manutenção e evolução do dicionário de dados.

Na linguagem adotada, são identificadas cinco operações de gerência, sendo que a contribuição da linguagem SQL3 nestas operações se limitam a criação e exclusão de regras.

Nesta proposta adotada, foi importante incorporar outras operações**,** porém estas considerações devem e são analisadas com mais detalhes em capítulos posteriores.

3.5 CONCLUSÕES

O modelo de definição de regras adotado, melhorou a expressividade da definição de regras da linguagem SQL3 e, além disso, ampliou o conjunto de operações de gerência de regras, com outras operações. Estas operações de gerência formam uma linguagem de gerência de regras. Esta linguagem de gerência não é flexível o suficiente para propiciar ao usuário a manutenção necessária das regras armazenadas no banco de dados. Esta flexibilidade está relacionada com a facilidade em alterar as regras ou manipular suas partes constituintes, da mesma forma em que seja necessário eliminar regras ou parte delas. Por exemplo, regras ECA_1A_2 poderiam ser modificadas para regras ECA, por meio de uma operação de alteração da regra, na qual a ação secundária seria eliminada. O objetivo desta manutenção é propiciar a evolução do dicionário de regras, de tal forma que ele tenha mais uma opção para o armazenamento das regras de negócio de sua aplicação, além da própria aplicação.

Para que seja possível prover esta linguagem de gerência de regras com outras operações de gerência, é necessário analisar e definir um ambiente que suporte estas operações, o que não é sugerido no modelo adotado. Este ambiente compreende um repositório de regras específico para armazenar as regras, desvinculando-as do dicionário de regras. Esta separação entre regras e dados deve ser analisada e discutida. Além do repositório, é necessário analisar e especificar as necessidades de meta-regras para manter a consistência das regras armazenadas neste repositório de regras.

Este trabalho tem como objetivo estender as operações de gerência de regras e, portanto, propor um repositório de regras para armazenar os tipos 1 e 2, de regras, oferecidos na linguagem adotada, neste trabalho. Este repositório de regras é a base para que sejam sugeridas as operações de gerência de regras. No capítulo seguinte é analisado e proposto este repositório.

CAPITULO 4. PROPOSTA DE UM REPOSITÓRIO DE REGRAS

4.1 INTRODUÇÃO

Um repositório de regras consiste de um conjunto de estruturas inter-relacionadas para armazenar regras e um conjunto de meta-regras para que garantam a consistência das regras armazenadas nesse repositório. O principal objetivo de um repositório de regras é dar suporte ao armazenamento das regras, considerando os diferentes tipos descritos no meta-modelo, como também as informações semânticas associadas a cada regra, com a finalidade de facilitar a gerência de regras. Neste capítulo, apresenta-se o repositório de regras e os mecanismos utilizados para manter a consistência das regras nele armazenadas. Estes mecanismos são meta-regras que atuam quando regras são manipuladas por meio de operações de gerência, sendo que estas meta-regras cumprem o mesmo papel que as restrições de dados, gerenciadas pelo SGBDA para manter a consistência dos dados, no dicionário de dados.

Inicialmente são identificados os elementos das regras que serão especificados no repositório. Estes elementos são definidos a partir dos tipos de regras, de seus inter-relacionamentos e relacionamento destes tipos com os objetos do banco de dados.

Uma vez que os elementos estejam identificados, são apresentados os elementos da Linguagem SQL3 que são reutilizados nesta proposta. O conjunto resultante é tomado como referência para a elaboração do repositório de regras aqui proposto.

A seguir especifica-se o repositório de regras. Este repositório é representado por meio de um diagrama entidade-relacionamento, nas quais as entidades correspondem às partes constituintes das regras e também objetos do dicionário de dados, tais como tabelas e atributos. Os relacionamentos entre entidades representam o relacionamento das regras. Um exemplo deste relacionamento é a associação entre a ação de uma regra com o evento de outra regra, por meio de operações de manipulação de dados. A associação representa uma ou mais operações de banco de dados que são especificadas na ação da regra e que também é especificada no elemento evento de outras regras, para o mesmo repositório de regras.

Por último, são definidas regras que representam os dois tipos de regras adotados neste trabalho. Estas regras são armazenadas no repositório de regras obedecendo a determinados critérios de inserção. Estes critérios são apoiados pelas meta-regras, cujo resultado final, após

a inserção de todas as regras, é um repositório de regras consistente.

4.2 SÍNTESE DAS CARACTERÍSTICAS DAS REGRAS

As características das regras relevantes para este trabalho e que são consideradas para elaboração do repositório de regras são identificadas por meio de uma análise realizada sobre o modelo de regras adotado, e que foi descrito no capítulo anterior. Estas características são agrupadas em função de cada elemento da regra. Portanto, são apresentadas as características relativas ao evento, depois as características associadas à condição e finalmente as características relacionadas à ação da regra. Além das características associadas a cada elemento da regra, é possível identificar características que dizem respeito à regra, tais como o tempo de ativação e as variáveis de transição.

Além de apresentar tais características, também são identificadas as características pertencentes ao modelo de regras da linguagem SQL3, com a finalidade de ilustrar a evolução destas características no modelo de regras adotado. Esta evolução é importante porque a proposta deste trabalho é dar seguimento a esta evolução, propondo uma extensão da linguagem de gerência de regras. O ponto de partida é o modelo de definição de regras adotado. Para evidenciar as diferenças apresentadas pela linguagem SQL3 e a proposta adotada, realiza-se uma análise pontual de cada elemento da regra e, para cada um deles, são tomadas como base as características da linguagem SQL3 e, em negrito, são adicionadas as características da linguagem proposta. Portanto, o que está em negrito são extensões propostas sobre a linguagem SQL3. Por exemplo, o tempo de ativação (*BEFORE* e *AFTER*) está presente em ambas as linguagens e não deve aparece em negrito, porém o tempo de ativação (**ON**), que foi sugerido na linguagem proposta, é apresentado em negrito.

- Regra

  A regra é constituída das seguintes informações:

  - ✓ Tempo de ativação: {***ON***, *BEFORE* ou *AFTER*}

    A linguagem SQL3 apresenta dois tipos de tempo de ativação (*BEFORE* ou *AFTER*). Estes elementos são utilizados em conjunto com os eventos permitidos pela linguagem SQL3, que são as operações de manipulação de dados, *insert, update* e *delete*. O primeiro tempo de ativação *BEFORE* é

utilizado com as três operações de manipulação de dados e o segundo tempo de ativação, *AFTER*, é utilizado somente com as operações *insert, update.*

Na extensão adotada, sugere-se a adição de outro tipo de tempo de ativação, ***ON***, em virtude de outros tipos de eventos que são sugeridos. Este tempo de ativação é utilizado para as operações *select* e para eventos de tempo. A seguir é apresentado um exemplo de criação de regras adotando o tempo de ativação ***ON***.

```
CREATE RULE Auditoria
ON SELECT OF salario ON PROF
FOR EACH ROW
WHEN (current.status = 'Doutor')
DO …
```

- ✓ Variáveis de Transição {***current***, *Old*, *New*}

As variáveis de transição *old* e *new* são utilizadas para acessar os dados antigos e novos que são afetados pelos eventos, cujas operações são *Insert*, *Delete* e *Update*. Com a adição da operação *select*, é adicionada a variável de transição *current*, cuja finalidade é acessar os dados atuais que são consultados, conforme apresentado na regra anterior.

Além destas informações, variáveis de transição e tempo de ativação, uma regra possui também outros atributos, a saber: um identificador único para a regra, data de criação, status e definição da regra. Estes atributos são comuns às duas linguagens e são apresentados e analisados na seção 4.3, que trata sobre a descrição do repositório de regras.

- Evento

O meta–modelo de regras, especificado para as regras do tipo 1, apresentado na Figura 6, considera que um evento, obrigatoriamente, deve estar associado a uma ou mais regras, e que uma regra deve, obrigatoriamente, estar associada a um ou mais eventos. Esta obrigatoriedade é justificada pelo fato de que, na linguagem SQL3, o evento é definido em conjunto com a definição da regra. Este evento representa uma operação de banco de dados, seja ela uma operação de *update* ou *delete*, *insert* ou *select*. Da mesma forma, incluem-se

também as operações *logon*, *logoff*, e eventos de tempo, caracterizados por uma ocorrência absoluta em um determinado momento no tempo, definido pela data/hora do sistema.

Um evento pode ser definido em função de outros eventos, de tal forma que, em uma mesma sentença é possível descrever mais de uma operação, seja ela de banco de dados e meta-dados. A este evento, composto por dois ou mais eventos, é definido como evento composto. Este conceito foi apresentado e discutido amplamente em Chakravarthy e Mishra (1993), Chakravarthy et al. (1994), Geppert, Gatziu e Dittrich (1995), Paton (1998) e Vaduva (1999).

As regras do tipo 2, cuja base é constituída pelos elementos condição e ação, conforme é apresentado na Figura 6, são disparadas em função da operação *Fire*. Esta operação tem a responsabilidade de disparar a regra, a ela associada, sempre que for solicitada. A diferença entre esta operação, *Fire*, e as demais operações, utilizadas no evento de banco de dados, é que, para o primeiro caso, a regra é disparada por meio de uma declaração explícita, e para o segundo caso, o disparo ocorre de forma implícita, ou seja, existe disparo toda vez que duas situações acontecerem, a primeira situação é a ocorrência do evento, em uma regra hospedeira (regra que possui a definição da operação de banco de dados em uma de suas ações), e a segunda situação é a existência de uma regra que possua um evento especificado por meio da mesma operação. Neste sentido, o SGBDA deve contar com um mecanismo que detecte a presença do evento e dispare as regras que possuam o evento ocorrido.

- ✓ São considerados nas operações de gerência da linguagem proposta, descritas no capítulo 5, os eventos de banco de dados e eventos temporais, apresentados a seguir.

  Os eventos de banco de dados possuem um conjunto de operações que são subdivididas em três grupos: operações de banco de dados, operações sobre meta-dados e operações de sistema.

  - Eventos de Banco de Dados:
    - Operações de Banco de Dados {***Select***, *Insert*, *Delete* e *Update*};
    - Operações Meta-Dados {***Alter***, **Drop**, ***Rename***};
    - Operações Sistema {***Logon***, ***Logoff***};

  - Eventos Temporais

- Hora/Data { **dd/mm/aaaa ::hh:mm:ss**}

- Condição

A condição é composta de sentenças SQL conectadas por operadores {*AND* e *OR*}. Além dos operadores apresentados, é considerado também o operador unário {***NOT***}. Este operador é utilizado como forma de negação da sentença que é definida na condição.

- Ação

A ação pode conter sentenças procedimentais SQL que incluem operações de banco de dados, operações para chamadas a procedimentos (***CALL*** (nome_procedimentos) e funções, operações para chamadas a outras regras (***Fire*** (nome_regra)) e blocos SQL).

- Relacionamento entre regras

O relacionamento entre regras ocorre de duas maneiras. A primeira, em função da ocorrência de eventos, e a segunda em função da definição declarativa de uma operação de gerência. No primeiro caso, é necessário que exista a definição de uma operação de banco de dados, na ação de uma regra, chamada de regra hospedeira. Da mesma forma, é necessário que uma ou mais regras contenha em seu evento, o mesmo tipo de operação sobre o mesmo objeto de banco de dados, outrora definido na ação da regra hospedeira. Quando a operação, especificada na ação da regra hospedeira, for solicitada, seja por uma aplicação ou explicitamente por um usuário, esta regra desencadeia o disparo de outras regras. Para que este encadeamento ocorra, é necessário que o SGBDA identifique, para cada operação de banco de dados executada, quais são as regras que possuem um evento com estas características.

Este encadeamento pode ocorrer com um conjunto constituído de um número elevado de regras, produzindo mudanças no estado do banco de dados, que nem sempre são desejáveis. Considerando que não existem erros associados à execução deste encadeamento de regras, então fica difícil identificar o resultado de um processo de negócios de resultados que são anômalos, sem um devido rastreamento da execução destas regras.

O segundo caso difere do primeiro, no sentido de que a regra hospedeira contém a definição explicita da chamada a outra regra, por meio de uma operação definida pelo

usuário. Neste segundo caso, o encadeamento também esta presente.

- Composição de Regras

  A composição de regras ocorre quando uma regra contém, explicitamente, outras regras. A composição de regras tem a finalidade de solucionar o problema gerado pelo disparo de múltiplas regras, disparadas pelo mesmo evento. Quando regras são disparadas pelo mesmo evento, gera-se um conjunto de regras que deve ser executado em função de algum critério. A linguagem SQL3 sugere, como critério para a ordem de execução destas regras, o uso da função data/hora de criação delas. O conceito de **composição de regras** sugere o uso de uma operação de gerência, ***Fire,*** para disparar as regras definidas sob o mesmo evento. Desta forma o resultado é uma execução controlada de regras.

Considerando as informações apresentadas anteriormente, é possível identificar um conjunto de características que estendem o modelo de regras da linguagem SQL3.

A linguagem de regras adotada utiliza informações sobre os objetos que estão armazenados no dicionário de dados do banco de dados, da mesma forma que a linguagem SQL3. A diferença reside no fato de que o modelo de regras adotado possui mais recursos que a linguagem SQL3, e como conseqüência, exige mais informações do dicionário de dados. Estes objetos são meta-dados que armazenam informações sobre tabelas, atributos, usuários, procedimentos e funções.

4.3 DESCRIÇÃO DO REPOSITÓRIO DE REGRAS

O repositório de regras é formado por um conjunto de estruturas que são utilizadas para armazenar as informações pertinentes às regras. O modelo de dados adotado para projetar o repositório de regras é o modelo relacional e como conseqüência, todas as estruturas são tabelas.

4.3.1 Tabela Regra

A principal tabela, denominada **REGRA**, armazena as características de todos os tipos de regras definidas no meta-modelo de regras. Estas características são:

- Código da regra ou Id_regra
- Nome da regra
- Autor
- Data de criação
- Status {habilitado / desabilitado}
- Tipo de regra {ECA, EA, ECAA, CA, CAA, A}
- Tempo de ativação da regra {*on*, *before*, *after*}
- Granularidade da regra {*row*, *statement*}

O <código da regra> ou <Id_regra>é utilizado para identificar uma única regra dentre todas as regras armazenadas na tabela Regras. O <nome da regra> contém uma descrição que representa a regra no contexto do negócio, do ponto de vista do usuário. Este nome, geralmente, é o mesmo utilizado quando de sua especificação no negócio. O <autor> é uma informação adicional que pode ser utilizada para armazenar informações sobre a pessoa responsável pela regra, seja este, quem a especificou no sistema ou o idealizador da mesma. A <data de criação> é o momento, no tempo, em que esta regra é criada, ou seja, seu dia/mês/ano::hora/min/seg. O <status> representa um estado, habilitada ou desabilitada. O status "habilitado" representa um estado em que a regra pode se encontrar no repositório de regras, ou seja, este status habilita a regra ser disparada, uma vez que ocorra um evento. O status "Desabilitado" representa um estado em que a regra pode assumir em um dado momento, no repositório de regras, sendo que este status implica que a regra não pode ser disparada, a não ser que ela mude de status.

O <tipo de regra> identifica o tipo de regra que foi armazenada no repositório. Este atributo recebe somente valores que pertencem ao seu domínio, definido pelos valores (ECAA, ECA; EA; CAA; CA; A). O <tempo de ativação> representa o momento em que a regra será executada, em função da ocorrência do evento. O atributo <tempo_ativ> recebe somente valores definidos em seu domínio. Neste caso, os valores são (*ON, BEFORE, AFTER*). Quando não for especificado na definição da regra, o tempo de ativação é atribuído

de forma automática. O termo automático implica que o sistema gerenciador assume um valor previamente definido como valor padrão. A granularidade da regra representa o número de vezes que a regra é disparada em função do evento. Os valores pertencentes ao domínio do atributo <granularidade> são (*ROW, STATEMENT*).

4.3.2 Tabela Evento

As informações sobre os elementos da regra são armazenadas nas seguintes tabelas: tabela EVENTO, tabela CONDIÇÃO e tabela AÇÃO. A relação entre evento e regra ocorre da seguinte maneira: cada regra pode ser disparada por um ou mais eventos e um evento pode disparar uma ou mais regras. As informações que devem ser armazenadas sobre este relacionamento constam na tabela REGRA-EVENTO (Regra_ev). Os atributos da tabela **EVENTO** são.

- Código do evento ou Id_evento
- Tipo_evento (banco de dados, temporal, sistema, meta-dados, evento_usr)
- Operação {*insert*, *update*, *delete*, *select,* data_hora*, logon, logoff, alter, drop, rename, fire*}
- Identificação de usuário ou Id_usuario
- Identificação de data_hora ou Id_data_hora
- Identificação da tabela ou Id_tabela

O <código do evento> armazena a identificação de cada evento. Cada evento é único. O evento é criado em função da criação de uma regra. Quando uma regra é criada, o SGBDA identifica o evento que foi definido na especificação da regra e armazena este evento na tabela EVENTO. O <tipo_evento> contém informações sobre o tipo de evento que é armazenado na tabela EVENTO, este pode ser do tipo evento de banco de dados (*insert*, *update*, *delete*, *select*), evento temporal (data_hora), evento de sistema (*logon, logoff*), evento meta-dados (*alter, drop,rename*) e evento definido pelo usuário (*Fire*). O <Id_usuario> identifica o código do usuário definido em conjunto com o evento de sistema (exemplo: *logon* Maria). O <Id_data_hora> identifica a data e/ou hora definida para o evento temporal. Este dois atributos, Id_usuario e Id_data_hora, podem receber valores *null`s*.

O atributo <Id_tabela> identifica a tabela que sofre a ação da operação definida no

evento.

4.3.3 Tabela Regra-Evento

A tabela **REGRA-EVENTO**, representada pelo relacionamento **"Regra_Ev",** existe em função do tipo de associação entre a ocorrência de regra e ocorrência de evento. Uma regra está relacionada, obrigatoriamente, com um ou mais eventos, portanto para cada regra, definida na tabela REGRA, deve existir o seu correspondente valor na tabela REGRA-EVENTO. Um evento definido na tabela EVENTO, necessariamente deve existir na tabela REGRA-EVENTO e pode estar associada a mais de uma regra. Para as regras que possuem evento composto, para cada evento existente na regra, existe uma ocorrência na tabela EVENTO. A regra que possui correlação com este evento composto, apresenta tantas ocorrências na tabela REGRA-EVENTO, quanto o número de eventos definidos no evento composto. Os atributos da tabela REGRA-EVENTO são:

- Código da regra ou Id_regra
- Código do evento ou Id_evento

Estes dois atributos, em conjunto, definem uma ocorrência de uma regra com seu respectivo evento. Na Figura 12 é apresentado o diagrama entidade-relacionamento referente ao relacionamento entre a entidade REGRA e a entidade EVENTO.

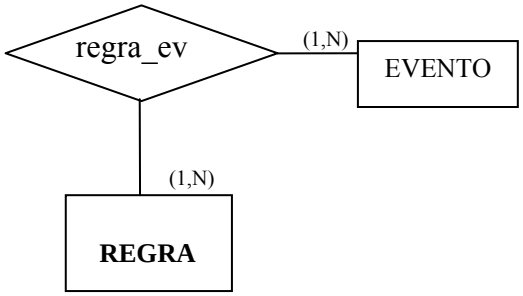

Figura 12 - Representação do relacionamento entre regra e evento.

4.3.4 Tabela Condição

Os elementos condição e ação da regra têm sua existência condicionada à existência da regra. A condição e a ação devem ser definidas a partir da definição da regra. Na Figura 13 é apresentado o diagrama entidade-relacionamento referente ao relacionamento da regra com os elementos condição e ação.

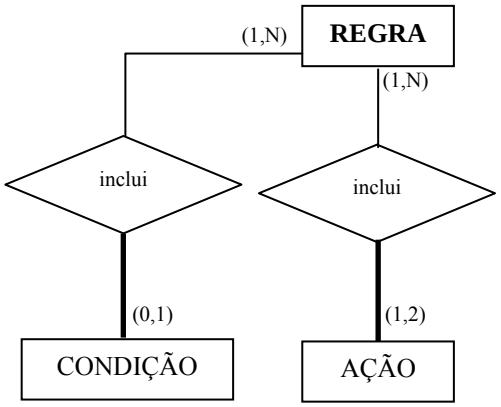

Figura 13 - Representação do relacionamento entre regra e os demais elementos condição e ação.

As informações que são consideradas na tabela **CONDIÇÃO** são:

- Código da regra ou Id_regra
- Código da condição ou Id_ condição
- Definição da condição
- Data_última_modificação

O <código da regra> é um atributo que possui os mesmos valores anteriormente definidos para a identificação de uma regra, na tabela REGRA. O <código da condição> identifica uma única condição. Ambos os atributos identificam a relação de pertinência da condição na regra. Quando uma regra é disparada, é possível identificar qual é a condição que deve ser avaliada. Uma condição pode ser reutilizada e ser atribuída a uma outra regra. Esta tarefa é possível por meio de uma operação de gerência que realize esta atribuição. Existem situações que devem ser analisadas, pelo analista, com a finalidade de garantir a correta

atribuição de uma condição a uma regra. A primeira situação ocorre quando a regra apresenta variáveis de transição, e estas variáveis devem ser especificadas na condição, sejam elas, variáveis locais ou globais. A segunda situação ocorre quando da atribuição de uma condição, que possui variáveis locais ou globais, a uma regra. Em ambos os casos, fica a cargo do analista efetuar os devidos ajustes. A "Data_última_modificação" é um atributo utilizado para armazenar informações sobre o momento em que a condição foi modificada por alguma operação de gerência associada à modificação da condição da regra.

4.3.5 Tabela Ação

Uma regra possui obrigatoriamente uma ação, porém existe a possibilidade de que uma outra ação seja definida em função do tipo de regra especificado. A ação primária é obrigatória, a outra ação, definida como ação secundária é opcional. Ambas as ações são armazenadas na tabela AÇÃO. Os atributos desta tabela, **AÇÃO**, são:

- Código da regra ou Id_regra
- Código da ação ou Id_ ação
- Categoria {*primaria*, *secundaria}*
- Data_ultima_modif
- Definição da ação

O <código da regra> é um atributo que contém os mesmos valores armazenados no atributo <código da regra> da tabela REGRA. O <código da ação> representa uma ação. Ambos os atributos são utilizados para identificar a relação de pertinência da ação na regra. Nesta tabela, o código da regra pode ser repetido, no máximo, duas vezes. Uma vez para cada tipo de ação. A <categoria> identifica o tipo de ação que foi armazenada. Neste sentido uma ação pode receber o valor "secundária", se e somente se, a mesma regra está associada a uma ação cuja categoria é primária. A <definição da ação> contém a especificação da ação da regra, ou seja, o corpo da ação, composto de código. Uma ação pode ser eliminada ou adicionada, em função do tipo de regra especificado na tabela, por meio de operações de gerência. É função do analista garantir que as variáveis de transição e variáveis locais e globais sejam adequadas ao contexto da regra, sempre que for necessário. A "Data_última_modificação" é um atributo utilizado para armazenar informações sobre o momento em que a ação foi modificada por alguma operação de gerência, associada à modificação da ação da regra

Na ação de uma regra, é possível definir operações de banco de dados, chamadas a procedimentos e execução de regras por meio da operação *Fire*.

4.3.6 Tabela Ação_EV

Regras que apresentam operações de banco de dados podem disparar outras regras, sempre que exista alguma regra cuja operação definida no evento é a mesma operação definida na ação da regra. Identificar a existência destas operações na ação, por meio de operações de gerência, é um meio eficaz de identificar encadeamentos de regras.

Existem dois tipos de encadeamentos: encadeamentos controlados e encadeamentos não controlados. Os primeiros ocorrem em função da especificação do negócio, na qual o analista define um conjunto de regras de negócio por meio de um encadeamento. O resultado final da execução deste conjunto de regras encadeadas é um resultado significativo e observável.

O segundo tipo de encadeamento ocorre em função de regras que são armazenadas no repositório de regras e que são disparadas em função da execução de outras regras, este encadeamento não está associado aos processos de negócio, mas sim a execuções que ocorrem, em função de outros processos. Este tipo de encadeamento de regras pode levar à execução infinita de um conjunto de regras. Para identificar o encadeamento de regras, é criada uma tabela **AÇÃO_EV** que armazena as operações de banco de dados existentes na ação de uma regra e também armazena a identificação dos eventos que possuem estas mesmas operações.

Na Figura 14 é ilustrada a tabela AÇÃO_EV

Figura 14 - Representação do relacionamento entre ação e evento.

Os atributos da tabela AÇÃO_EV são:

- Código da regra ou Id_regra

- Código da ação ou Id_ ação
- Id_evento

O <código da regra> e o <código da ação> identificam uma ocorrência do elemento ação. Nesta ação, existem operações de banco de dados que podem gerar um encadeamento de regras. Estas operações fazem parte de eventos que são definidos na tabela EVENTO. Da mesma forma, este encadeamento pode ocorrer por meio da operação *Fire*. A tabela AÇÃO_EV tem a finalidade de rastrear este encadeamento, identificando quais são as regras que podem disparar outras regras, porém o disparo destas regras ocorre em função de algum critério previamente estabelecido. O critério adotado é o mesmo apresentado pela linguagem SQL3.

A linguagem SQL3 sugere uma operação de gerência para criar e para eliminar regras, porém, na prática, os SGDBA comerciais implementam a operação de criação ou atualização de regras em um mesmo comando, a saber: "*create or replace trigger*". Quando este comando é utilizado para atualizar regras, é gerada uma nova data de criação para a regra modificada, e como conseqüência, estabelece uma nova seqüência de execução de regras. Para o analista, o fato de modificar a ordem de execução, pode comprometer a execução de um processo de negócio.

4.3.7 Tabela Regra_Composição

O elemento "ação" de uma regra pode conter operações do tipo *Fire*. Esta operação tem a função de executar, explicitamente, regras do tipo "condição-ação". Isto implica que uma "ação" pode ser composta por uma ou mais definições explícitas de disparo de regras. O fato de que, uma operação de disparo é definida explicitamente sobre uma regra, na ação de uma regra, considera-se, então, que existe uma composição de regras. Assim, uma regra pode ser composta por mais de uma regra e cada uma destas regras pode fazer parte de mais de uma ação de uma regra. Além disso, cada composição, por sua vez, pode conter outras regras do tipo 1.

Este encadeamento de regras é materializado por meio de uma tabela que contém atributos que identificam as regras e suas respectivas prioridades de execução. Esta tabela é definida como REGRA_COMP ou **REGRA-COMPOSIÇÃO**. Os atributos desta tabela são:

- Código da regra ou Id_regra

- Código da ação ou Id_ ação
- Composta_de
- Prioridade

O atributo <código da regra> identifica a existência de uma regra. O <código da ação> identifica a ação, primária ou secundária, que possui um conjunto de regras. O atributo <composta_de> armazena todas as regras do tipo condição-ação definidas na ação. Estes três atributos <código da regra> <código da ação> e <composta-de> identificam uma única ocorrência de regra definida na ação de uma regra. O atributo <prioridade> armazena a prioridade de execução das regras. Estes valores de prioridade são números inteiros ascendentes (1,2,3...,n) que representam a ordem de execução da regra. Quanto menor o número, maior o grau de prioridade. Quando ocorre a modificação de uma das regras definidas na ação, por meio de uma operação de gerência, a ordem de execução de regras acompanha esta mudança. Na Figura 15 é apresentado o diagrama entidade-relacionamento para a representação da relação entre a tabela REGRA e a tabela REGRA-COMPOSIÇÃO.

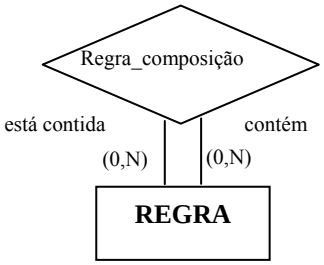

Figura 15 - Representação entidade-relacionamento de uma composição de regras.

A característica principal do modelo de regras adotado é que as regras são representadas de forma independente do conjunto de tabelas do banco de dados. Esta independência implica em que uma regra pode ser dotada de um evento que não esteja relacionado com os objetos de banco de dados, como tabelas e colunas. De forma oposta, na linguagem SQL3, uma regra, ao ser especificada, deve, obrigatoriamente, estar associada a alguma tabela do banco de dados, haja vista pelo tipo de evento de banco de dados permitido (*insert, update* ou *delete*).

Quando a especificação do evento de uma regra contempla a existência de um meta-dado, então é necessário que se verifique a existência deste meta-dado, no dicionário de dados, antes de armazenar a regra no repositório de regras. A seguir é apresentada a

associação entre os elementos da regra: evento, condição e ação com os objetos do banco de dados.

4.3.8 Tabelas Usuário_Sistema, Coluna_Sistema e Tabela_Sistema

A tabela EVENTO está associada com as tabelas COLUNA_SISTEMA, TABELA_SISTEMA e USUARIO_SISTEMA que fazem parte do **dicionário de dados**. A tabela **COLUNA_SISTEMA** representa as colunas da tabelas que podem ser definidas em um evento, quando o evento for um evento-dado. O relacionamento **EV_COL** entre a tabela **COLUNA_SISTEMA** e a tabela **EVENTO** ocorre por meio do relacionamento **REGRA_EV**. Este relacionamento contém informações sobre as colunas utilizadas no evento de uma regra. A **TABELA_SISTEMA** representa as tabelas do banco de dados, que também podem ser utilizadas no evento-dados, relativas ao conjunto de eventos de banco de dados, conforme descrito no item 3.3.1. A tabela **USUARIO_SISTEMA** armazena informações sobre os usuários do banco de dados e estes dados são utilizados no evento sistema.

A associação entre o evento e as tabelas do sistema (coluna, tabela e usuário) é do tipo não obrigatória, por esta razão é possível definir um evento-temporal ou evento-sistema de forma que se mantenha a independência de dados. Na Figura 16 são apresentadas as associações entre o evento e os demais objetos do banco de dados. A entidade regra_ev é uma entidade associativa (HEUSER, 2004). Sendo regra_ev uma entidade, esta pode ser associada a COLUNA_SISTEMA, por meio do relacionamento ev_col.

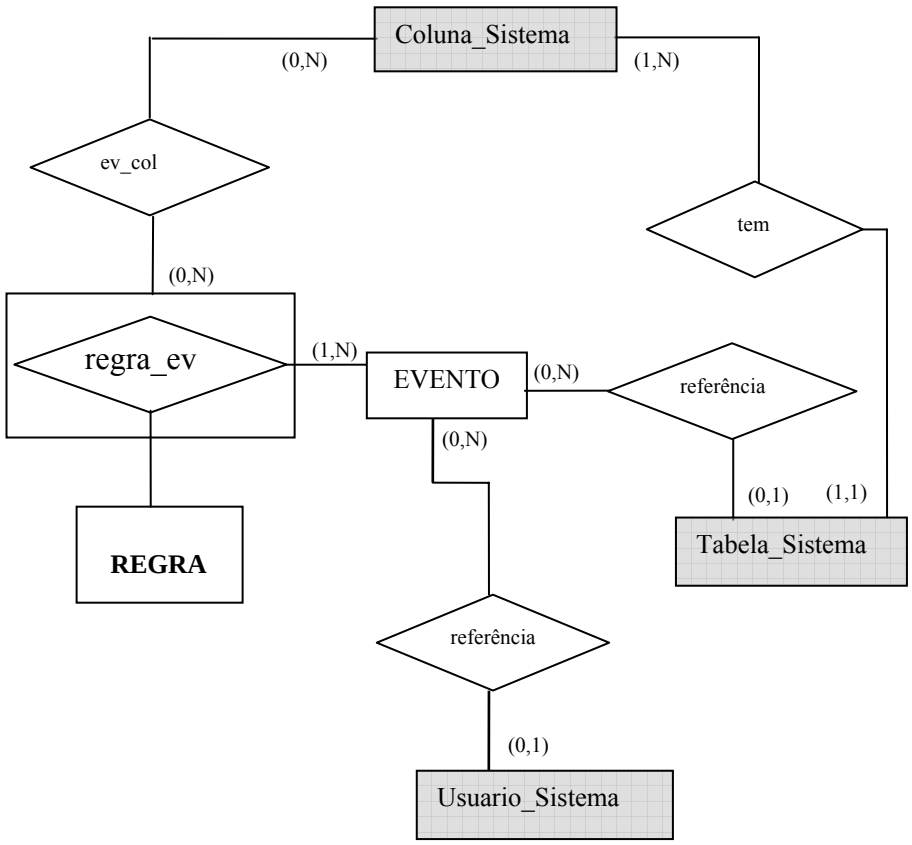

Figura 16 - Diagrama entidade-relacionamento da associação entre o evento e os demais objetos do banco de dados.

Os elementos da tabela **EV_COL** são:

- Código do evento ou Id_evento
- Código da coluna ou Id_coluna

Esta tabela, **EV_COL**, armazena as informações sobre as colunas utilizadas no evento. Um evento pode fazer referência a uma tabela e cada tabela pode ser referenciada por mais de um evento.

O elemento condição e o elemento ação também utilizam as informações contidas na tabela TABELA_SISTEMA. Na Figura 17 são apresentados os relacionamentos pertinentes à ação e à condição com TABELA_SISTEMA.

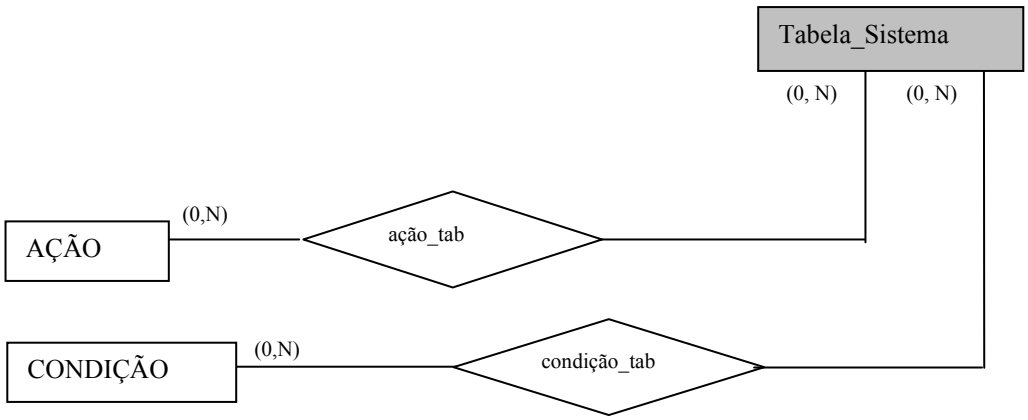

Figura 17 - Diagrama entidade-relacionamento da associação entre a TABELA_SISTEMA e as tabelas condição e a ação.

A tabela **AÇÃO_TAB** armazena informações sobre as tabela utilizadas na ação da regra. Seus atributos são:

- Código do regra ou Id_regra
- Código da ação ou Id_ação
- Código da tabela ou Id_tabela

A tabela **CONDIÇÃO_TAB** armazena informações sobre as tabelas que são utilizadas na condição da regra. Os elementos desta tabela são:

- Código do regra ou Id_regra
- Código da ação ou Id_condição
- Código da tabela ou Id_tabela

O repositório de regras compreende um conjunto de tabelas e um conjunto de meta-regras. As tabelas são utilizadas para armazenar as regras, seus inter-relacionamentos e composição de regras. As meta-regras têm a função de garantir a consistência do repositório de regras. Elas são exigidas sempre que são utilizadas operações de gerência para definir, excluir ou alteradas as regras.

Para validar o repositório de regras proposto, primeiro são apresentadas as estruturas

deste repositório e logo a seguir, um conjunto de meta-regras, utilizadas para manter sua consistência. As tabelas do repositório de regras são apresentadas a seguir. São descritas apenas as tabelas relevantes para o estudo de validação do repositório, no esquema lógico a seguir, visto que as demais tabelas que fazem parte do dicionário de dados, do banco de dados, não fazem parte da análise deste trabalho. A chave primária está sublinhada e chaves estrangeiras estão em itálico

REGRA (Id_regra, nome, autor, dataHora_criação, status, tipo_regra,
tempo_ativação, granularidade)

EVENTO (Id_evento, operação, tipo_evento, Id_usuario, datahora, Id_tabela)

REGRA_EVENTO (*Id_Regra*, *Id_evento*)

EV_COL (*Id_Regra*, *Id_evento*, *Id_coluna*)

CONDIÇÃO (*Id_regra*, Id_condição, definição_condição, data_ultima_modif)

CONDIÇÃO_TAB (*Id_regra, Id_condição, Id_tabela)*

AÇÃO (*Id_regra*, Id_ação, categoria, data_ultima_modif, definição_ação)

AÇÃO_TAB (*Id_regra, Id_ação, Id_tabela*)

AÇÃO_EV (*Id_regra*, *Id_ação*, *Id_evento*)

REGRA_COMP (*Id_regra, Id_ação*, *composta_de*, prioridade)

O esquema lógico descreve os atributos associados ao conjunto de tabelas do repositório de regras. A área delimitada pelo pontilhado, na Figura 18, representa o conjunto de tabelas que fazem parte do repositório de regras. As tabelas COLUNA_SISTEMA, TABELA_SISTEMA e USUÁRIO_SISTEMA fazem parte do dicionário de dados.

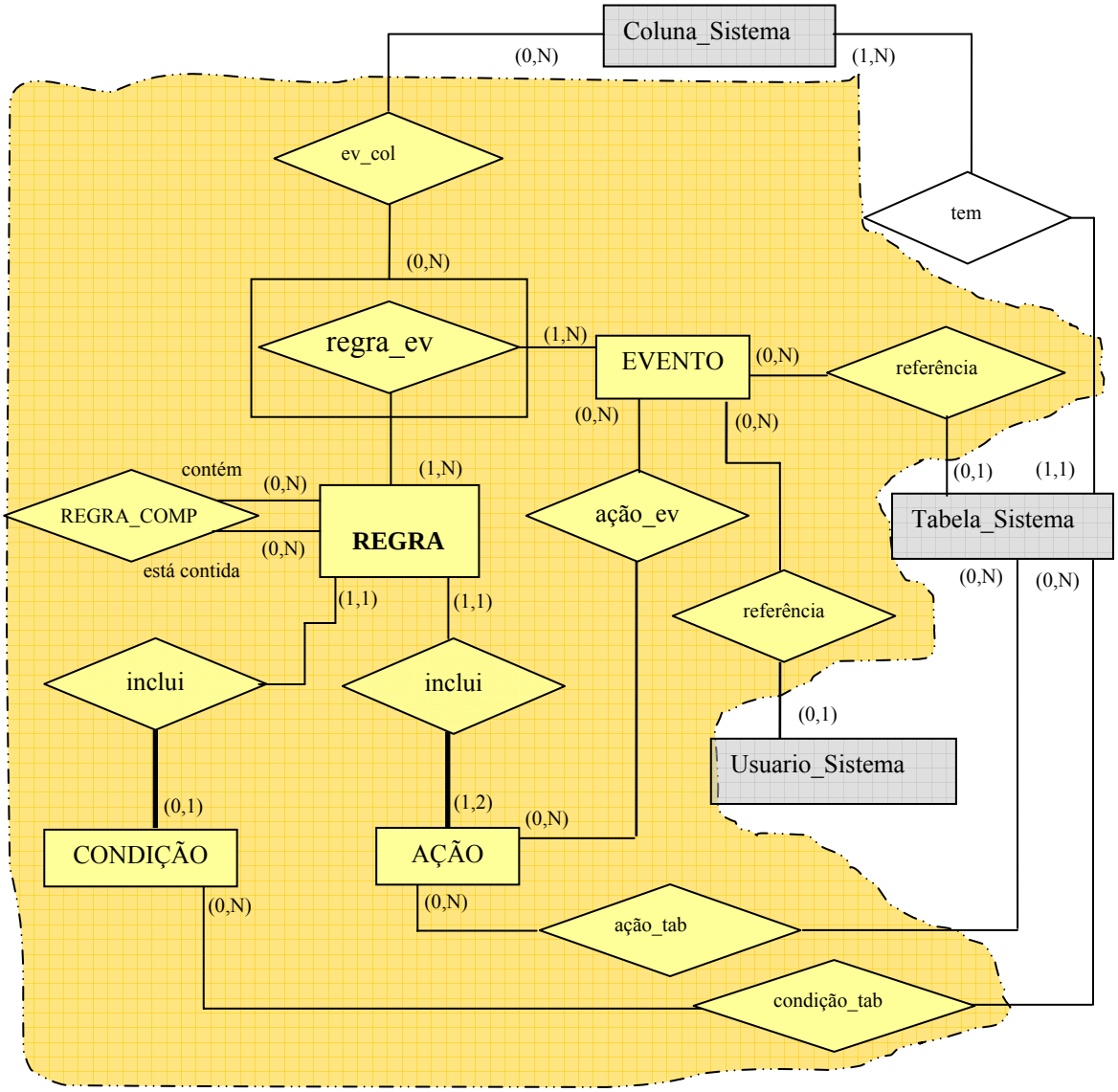

Figura 18 - Diagrama entidade-relacionamento relativo ao repositório de regras.

4.4 META-REGRAS DO DICIONÁRIO DE REGRAS

Um repositório de regras contém um conjunto de estruturas para armazenar regras, ao serem inseridas neste repositório, a execução das regras seguem uma ordem previamente definida. Esta ordem é descrita por meio de um conjunto de atividades apresentadas no diagrama de atividades ilustradas na Figura 19.

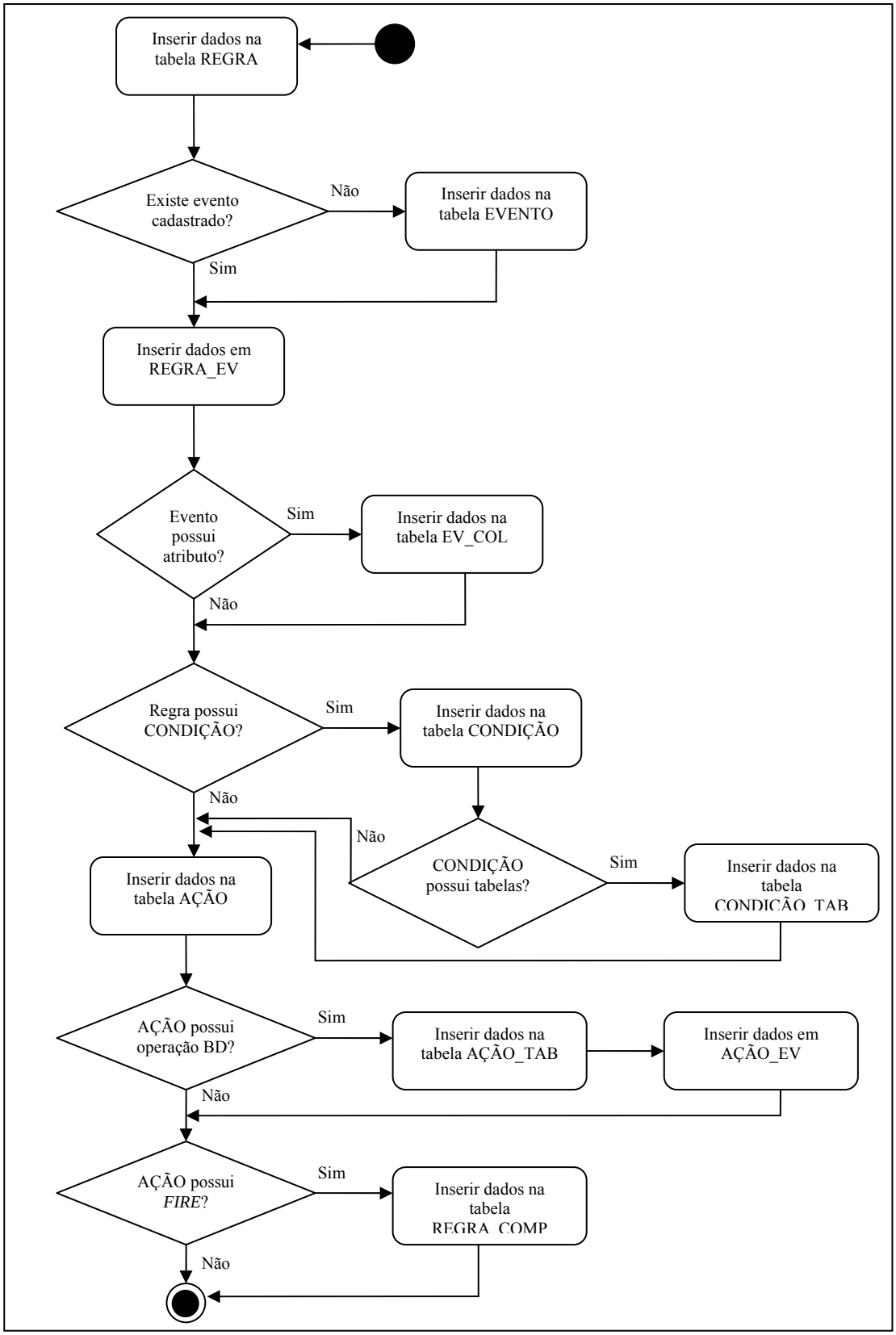

Figura 19 - Diagrama de atividade para a inserção de uma regra no repositório de regras.

Todas as atividades de inclusão de uma regra no repositório de regras partem do mesmo ponto inicial, a inserção de dados na tabela REGRA. Portanto, para a inserção das informações referentes aos elementos de uma regra, no repositório de regras, primeiro insere-se dados na tabela REGRA, logo a seguir, verifica-se a existência do evento, pertencente à regra, na tabela EVENTO. Caso o evento a ser inserido, já exista na tabela EVENTO, armazena-se apenas a associação deste evento à ocorrência da nova regra, caso contrário, armazenam-se tanto dados de evento quanto dados da associação deste evento com a regra.

O passo seguinte é verificar se o evento inserido possui atributos, caso afirmativo, inserem-se os dados na tabela EV_COL, caso negativo, verifica-se a existência do elemento condição na regra.

Quando a regra possui condição, os dados sobre a condição são armazenados na tabela CONDIÇÃO, e se por ventura, esta ação referência tabelas do banco de dados, armazena-se esta referência na tabela CONDIÇÃO_TAB.

A atividade seguinte é inserir dados sobre a ação da regra. Verifica-se a existência de tabelas na ação da regra, caso exista, inserir dados na tabela AÇÃO_TAB, e logo a seguir na tabela AÇÃO_EV. Os dados desta tabela são conseqüência da análise realizada sobre dois elementos da regra, a ação e o evento. Quando operações de banco de dados definidas na ação da regra são iguais as operações de banco de dados definidas no evento, então, armazena-se esta informação da tabela AÇÃO_EV.

Por último, verifica-se a existência da operação *Fire* na ação da regra. Caso existam, armazenam-se as regras que são disparadas por esta operação.

As meta-regras relacionadas com as atividades apresentadas no diagrama de atividades, da Figura 19, são apresentadas na Tabela 4.1. Nesta tabela são descritas sete meta-regras, que auxiliam a manter a consistência do repositório de regras. Estas meta-regras serão utilizadas pelo SGBDA toda vez que uma regra é inserida no repositório de regras.

Tabela 4.1 – Meta-regras associadas ao repositório de regras

Regra	Descrição
Meta-Regra 1	As Regras do tipo CA, CAA e A não devem possuir granularidade nem tempo de ativação
Meta-Regra 2	As Regras do tipo CA, CAA e A não podem utilizar variáveis de transição.
Meta-Regra 3	Somente as regras do tipo CA, CAA e A devem ter como evento a operação *Fire*
Meta-Regra 4	Somente as regras que têm como evento as operações *UPDATE*, *DELETE* e/ou *INSERT* possuem granularidade.
Meta-Regra 5	Somente as regras que possuem o elemento condição podem ter uma ação secundária.
Meta-Regra 6	Somente as regras que tem por evento a operação *SELECT* podem utilizar a variável de transição *CURRENT*
Meta-Regra 7	Toda vez que uma regra for disparada por mais de um evento, estes eventos devem ser do mesmo tipo.

A Meta-Regra 1 é responsável por verificar se o tipo de regra, derivação ou operacional, a ser inserida na tabela REGRA, possui granularidade e tempo de ativação. Da mesma forma, a Meta-Regra 2 verifica se estes tipos não possuem variáveis de transição. A Meta-Regra 3 verifica se o evento definido na regra tipo 2, a ser inserida na tabela REGRA é *Fire*. A Meta-Regra 4 verifica a existência de granularidade para regras que possuem as operações *UPDATE*, *DELETE* e/ou *INSERT*, definidas no evento. A Meta-Regra 5 verifica a existência do elemento condição na definição das regras que possuem duas ações, primária e secundária. A Meta-Regra 6 verifica se a variável de transição *CURRENT* é compatível com a operação de banco de dados definida no evento da regra. A Meta-Regra 7 verifica se os eventos compostos por mais de uma operação, são do mesmo tipo.

4.5 UM EXEMPLO PRÁTICO DO USO DO REPOSITÓRIO

Nesta seção, são apresentadas as tabelas do repositório de regras, e um exemplo prático de inserção de regra, neste repositório. Para isto, é utilizada, a título de exemplo, uma regra cujo evento, composto, tem a finalidade de disparar a regra em caso de uma inserção ou

atualização de uma reserva de equipamento. Esta regra é do tipo ECAA e representa a solicitação de um empréstimo de equipamento para uso em sala de aula. Parte-se da premissa de que o professor já efetuou uma reserva, e por necessidades de negócio, necessita atualizá-la. Seja uma solicitação de reserva ou atualização da mesma, o sistema deve verificar várias situações, que foram traduzidas para regras computacionais e implementadas na regra R12: a primeira regra verifica se o produto está disponível, logo depois, na segunda regra, se o professor está habilitado para realizar o empréstimo, a seguir, na terceira regra, se o professor obedece aos prazos mínimos para realizar a atualização e se tudo estiver aprovado, na quarta regra, é verificado o estoque, se este foi atualizado, em função desta nova reserva. Portanto, a regra R12 compõe-se de outras regras, que implementam estas verificações. A regra 12 é ilustrada na Figura 20. Esta regra é do tipo ação (A), e é composta de quatro regras disparadas pela operação *Fire*.

```
CREATE RULE R12
DO
 BEGIN
                FIRE R1;   /* VERIFICA A DISPONIBILIDADE DO PRODUTO. */
                FIRE R2;   /* VERIFICA SE O PROF. ESTA HAB/DESAB. */
                FIRE R3;   /* vERIFICA PRAZO DE 24 HORAS E DE 15 DIAS */
                FIRE R5;   /* ATUALIZA ESTOQUE        */
 END;
```

Figura 20 – Regra composta (R1, R2, R3, R5)

Uma vez que o professor realizou a atualização da reserva, uma mensagem é enviada a este, sinalizando que está tudo ‘ok’. Caso contrário, o professor é notificado de que a reserva não foi possível e um histórico é gravado deixando constância de que o professor foi notificado da inviabilidade de atualização da atualização de reserva.

Na Figura 21 é ilustrada a regra R8. Esta regra apresenta um conjunto de variáveis globais definidas pelos comandos DECLARE. Estas variáveis são utilizadas pela regra R12, contida na ação primária de R8.

```
/*variáveis globais*/
DECLARE v_codprof      numeric;
DECLARE v_codReserva numeric;
DECLARE v_codEquip    varchar2(10);
DECLARE v_quant        numeric;
DECLARE v_dataRes     date;
DECLARE v_horaIni      date;
DECLARE v_horaFim     date;

CREATE RULE R8
AFTER INSERT OR UPDATE OF cod_reserva ON RESERVA
FOR EACH ROW
WHEN ((SELECT   statusRes
          FROM     RESERVA
          WHERE    codReserva = :v_codReserva) = 'ATIVA')
DO
   BEGIN

        FIRE R12;        /* VERIFICA se o empréstimo do equip. é possível */
        CALL PROCEDURE COMUNICA_CLIENTE('INSERÇÃO OU ATUALIZAÇÃO _OK_');

    END;

ELSEDO
    BEGIN

        UPDATE CLIENTE
        SET       notificação = 'Sim'
        WHERE  codprof = v_codProf;

        CALL PROCEDURE COMUNICA_CLIENTE('INSERÇÃO OU ATUALIZAÇÃO
                                                    NÃO PERMITIDA');

    END;
```

/* Estas variáveis são utilizadas também dentro da regra **R12** */

Figura 21 – Especificação da Regra R8

Para inserir uma regra no repositório de regras, segue-se os passos apresentados no diagrama de atividades, ilustrado na Figura 19.

Considerando as regras R8 e R12, e assumindo que o usuário utiliza uma operação de gerência para inseri-las no repositório de regras, então a seqüência de inserção dos dados das regras é a seguinte: (antes do inicio do processo de inserção de uma regra, assume-se que foram realizadas análises sintática e semântica)

Parte-se da premissa que o repositório de regras está vazio. Os dados da regra R8, referentes ao nome da regra, tipo de regra, granularidade, tempo de ativação da regra são

inseridos no primeiro registro, da tabela REGRA, ilustrada na Tabela 4.2. Além destes dados, o SGBDA identifica o autor da regra, assume o tempo de criação da regra e atribui o status de habilitada. O mesmo ocorrendo com a regra R12, por meio da qual é inserido o segundo registro. Como a regra R12 é to tipo ação, ela não possui granularidade e tempo de ativação.

Tabela 4.2 - Tabela REGRA

ID_REGRA	NOME	AUTOR	DATA_CRIACAO	STATUS	TIPO_REGRA	GRANULARIDADE	TEMPO_ATIVAÇÃO
R8	Inserção ou Atualização Reserva	João Silva	4/3/06 11:00	H	ECAA	ROW	AFTER
R12	Pré-requisito Empréstimo	João Silva	4/3/06 10:50	H	A		

A próxima etapa é a inserção dos eventos, das regras R8 e R12. Na Tabela 4.3 são especificados os atributos da Tabela EVENTO. O atributo <Id_usuario> é utilizado para armazenar a identificação do usuário, definido em um evento do tipo evento sistema. O atributo <Data_hora> é utilizado para armazenar a data e/ou hora para os eventos do tipo evento temporal. A regra R8 possui dois eventos do tipo evento dados, a saber: *update* reserva e *insert* reserva. Cada evento é armazenado em um registro. Além destes eventos, na ação primária desta regra, é definido um evento do tipo *Fire*. Este evento é armazenado em outro registro da tabela EVENTO.

Tabela 4.3 - Tabela EVENTO

ID_EVENTO	OPERACAO	TIPO_EVENTO	ID_USUARIO	DATA_HORA	ID_TABELA
EV_8	UPDATE	BANCO DADOS			RESERVA
EV_8a	INSERT	BANCO DADOS			RESERVA
EV_12	FIRE	DEF_USUARIO			

O passo seguinte é correlacionar estes eventos a suas respectivas regras, armazenadas na tabela REGRA. Para tanto, utiliza-se a tabela REGRA-EVENTO, descrita na Tabela 4.4. Cada registro desta tabela representa uma associação entre uma regra e seu evento. Nesta tabela é representado o relacionamento de uma regra, R8, com mais de um evento, EV_8 e EV_8a.

Tabela 4.4 – Tabela REGRA_EVENTO

ID_REGRA	ID_EVENTO
R8	EV_8
R8	**EV_8a**
R12	EV_12

Quando o evento possui uma operação do tipo update ou select, que envolva atributos, então armazena-se estes atributos, associados a esse evento, na tabela EV_COL, conforme ilustrado na Tabela 4.5.

Tabela 4.5 – Tabela EV_COL

ID_REGRA	ID_EVENTO	ID_COLUNA
R8	EV_8	cod_reserva

Um evento pode estar associado a mais de um atributo e um atributo pode fazer parte em mais de um evento.

Considere que exista uma outra regra, R99, do tipo EA, cujo evento é *insert* reserva. Esta regra ao ser armazenada no repositório de regra, constituiria mais um registro na tabela REGRA, conforme ilustrado na Tabela 4.6.

Tabela 4.6 - Tabela EVENTO

ID_REGRA	NOME	AUTOR	DATA_CRIACAO	STATUS	TIPO_REGRA	GRANULARIDADE	TEMPO_ATIVAÇÃO
R8	Inserção ou Atualização Reserva	João Silva	4/3/06 11:00	H	ECAA	ROW	AFTER
R12	Pré-requisito Empréstimo	João Silva	4/3/06 10:50	H	A		
R99	**Inserção Reserva**	**Pedro Silva**	**4/4/06 12:50**	**H**	**EA**	**ROW**	**BEFORE**

Este evento é igual a um dos eventos, definidos na tabela EVENTO. Neste caso, por um lado, não é necessário inserir outro registro na tabela EVENTO, visto que esta informação já existe, por outro lado, um evento pode fazer referência a mais de uma regra da mesma forma que uma regra pode estar associada a mais de um evento. O relacionamento da regra R99 com o evento inserir reserva, é ilustrado na tabela 4.7, por meio da adição de outro registro.

Tabela 4.7 – Tabela REGRA_EVENTO

ID_REGRA	ID_EVENTO
R8	EV_8
R8	**EV_8a**
R12	EV_12
R99	**EV_8a**

O passo seguinte é a inserção da condição da regra. Este elemento é inserido na tabela CONDIÇÃO, ilustrado na Tabela 4.8

Tabela 4.8 - Tabela CONDIÇÃO

ID_REGRA	ID_CONDICAO	DATA_ULT_MODIF	DEF_CONDICAO
R8	COND8		Código

O atributo def_condição possui o código referente à condição da regra. Quando a condição é alterada, por meio de uma operação de gerência, por exemplo, o momento em que houve esta modificação, portanto a data de modificação, é armazenada no atributo data_ult_modif. Considerando que a condição contenha referências a tabelas do banco de dados, então, estas informações são armazenadas na tabela CONDIÇÃO_TAB, ilustrada na Tabela 4.9.

Tabela 4.9 - Tabela CONDIÇÃO_TAB

ID_REGRA	ID_CONDICAO	ID_TABELA
R8	COND8	RESERVA

O passo seguinte é o armazenamento das informações referentes à ação da regra. Na tabela 4.10 é apresentada a tabela AÇÃO, utilizada para armazenar as informações referenciadas na ação da regra.

Tabela 4.10 - Tabela AÇÃO

ID_REGRA	ID_ACAO	CATEGORIA	DATA_ULT_ MODIF	DEFINIÇÃO_ACAO
R8	A8P	PRIMARIA		Código
R8	A8S	SECUNDARIA		Código
R12	A12P	PRIMARIA		Código

A tabela AÇÃO possui dois registros relacionados à ação da regra R8. Esta regra possui duas ações, uma primária e uma secundária. A regra R12 possui somente uma ação

primária. O atributo definição_ação possui o código pertinente à ação da regra. Quando a ação da regra possui, em sua especificação, referências a tabelas do banco de dados, então, esta informação é armazenada na tabela AÇÃO_TAB, conforme é ilustrado na tabela 4.11.

Tabela 4.11 - Tabela AÇÃO_TAB

ID_REGRA	ID_AÇÃO	ID_TABELA
R8	A8S	CLIENTE

Existem situações nas quais a ação de uma regra contém operações de banco de dados que são as mesmas utilizadas em eventos de outras tabelas. Para identificar a ocorrência destes casos, estas informações são armazenadas na tabela AÇÃO_EV. A finalidade desta tabela é identificar as regras que podem disparar outras regras, por meio de operações de banco de dados. Considere que a regra R99, do tipo EA, possui em sua ação uma operação de banco de dados do tipo:

```
UPDATE RESERVA
SET cod_reserva = :v_cod_reserva;
```

Esta operação é a mesma operação definida na ação da regra R8. Portanto a regra R99, a ser executada, dispara a regra R8. Este rastreamento é apresentado na Tabela 4.12.

Tabela 4.12 - Tabela AÇÃO_EV

ID_REGRA	ID_AÇÃO	ID_EVENTO
R99	A99P	EV8

A última etapa é o armazenamento da composição de regras. A regra R12 possui em sua ação um conjunto de regras disparadas pela operação *Fire*. A composição de regras é armazenada na tabela REGRA_COMP, ilustrada na Tabela 4.13. O atributo prioridade identifica a prioridade de execução das regras, disparadas pela operação *Fire*. Esta prioridade indica a ordem em que estas regras, R1, R2, R3 e R5, estão dispostas na ação da regra. Quando duas regras são trocadas de ordem, então a prioridade é trocada automaticamente. Esta deveria ser uma operação automatizada pelo sistema.

Tabela 4.13 - Tabela REGRA_COMP

ID_REGRA	ID_AÇÃO	COMPOSTA_DE	PRIORIDADE
R8	A8P	R12	1
R12	A12P	R1	1
R12	A12P	R2	2
R12	A12P	R3	3
R12	A12P	R5	4

As regras, R1, R2, R3 e R5, deveriam ser armazenadas da mesma forma que as regras R8, R12 e R99 nas tabelas do repositório de regras. Elas não foram armazenadas porque o objetivo é utilizá-las, apenas, como referência, dentro da regra R12. Portanto, quando o repositório estiver em operação, a tabela REGRA deveria conter sete regras, a saber: R1, R2, R3, R5, R8, R12 e R99.

4.6 CONCLUSÕES

Neste capítulo foram identificadas as características dos tipos de regras que serão adotados na linguagem de gerência de regras. Estas características formam a base do modelo de definição de regras adotado. Além desta base, são reutilizados os conceitos sugeridos na própria linguagem SQL3 de regras. Este conjunto de características resultante, é o ponto inicial para a elaboração de um repositório de regras. Este repositório é composto de um conjunto de estruturas inter-relacionadas que são utilizadas para armazenar as regras, o relacionamento entre elas e o relacionamento delas com os demais objetos do banco de dados, com tabelas e atributos.

Neste trabalho são apresentadas e discutidas associações implícitas e explicitas entre regras. As associações explícitas ocorrem em função da composição de regras. Esta composição é representada por meio da declaração da operação *Fire* na ação da regra. A associação implícita, é representada por meio da definição de operações de banco de dados, na ação da regra. Quando estas operações são executadas pelo SGBDA, elas podem inicializar um encadeamento de regras. A implementação de regras em um SGBDA, pode levar a estados não controláveis, como por exemplo, conjunto de regras que se disparam entre si, gerando um ciclo infinito de execução de regras. Também pode levar a estados inconsistentes do banco de dados. Esta inconsistência é fruto de execuções de conjuntos de regras, não

desejáveis do ponto de vista do negócio, que foram executadas implicitamente, e não representam os processos de negócio desejados. Nos SGBDAs comerciais, existe um tratamento, pouco eficiente, para deter este ciclo de execução de regras, por meio de um corte abrupto na execução do ciclo de execução. Neste repositório, este problema é tratado por meio de uma tabela que armazena o relacionamento entre regras, seja este relacionamento implícito ou explícito. Desta forma, este relacionamento contém a as informações necessárias para rastrear encadeamentos de regras.

Considerando que regras são compostas de vários elementos, então a alteração destes elementos pode gerar inconsistências no repositório de regras. Surge então a necessidade de elaborar um conjunto de meta-regras, cuja finalidade é verificar se as regras a serem alteradas ou excluídas obedecem aos critérios definidas pelas meta-regras. No capítulo seguinte são sugeridas, descritas e exemplificadas outras operações de gerência, além das operações apresentadas pela linguagem adotada neste trabalho, com a finalidade de prover meios para que o usuário tenha condições de alterar as regras e também as suas partes.

CAPITULO 5. OPERAÇÕES DE GERÊNCIA DE REGRAS

5.1 INTRODUÇÃO

Uma linguagem de gerência de regras é composta de operações para a definição, exclusão e alteração de regras. A linguagem de Definição de regras adotada, define uma operação para a criação de regras.

Neste trabalho são apresentadas operações sobre regras. Estas operações estendem a linguagem de regras adotada e definem um novo conjunto de operações que caracterizam a linguagem de gerência de regras. Esta linguagem de gerência tem por base o repositório de regras. As operações da linguagem de gerência incidem sobre uma regra ou parte desta, a fim de alterá-la, consultá-la, eliminá-la, habilitá-la ou desabilitá-la.

As alterações das regras, sem os devidos mecanismos que garantam que tais alterações mantenham a consistência do repositório de regras, tal como se encontrava antes de inicializar tais modificações, pode levar este repositório a um estado inconsistente. Estes mecanismos são meta-regras cuja finalidade é garantir a consistência do repositório de regras. Neste trabalho, são elaboradas e analisadas meta-regras para dar suporte à gerência de regras. Cada operação de gerência é acompanhada de sua devida discussão, e quando necessário apresenta-se as meta-regras associada ao emprego da respectiva operação, e um exemplo ilustrativo.

5.2 PROPOSTA DE UMA LINGUAGEM DE GERÊNCIA DE REGRAS

Uma Linguagem de Gerência de Regras (LGR) é composta de um conjunto de operações que incidem sobre uma ou mais regras, de um repositório de regras. A linguagem SQL3 possui um conjunto de operações de gerência voltado para a definição e exclusão de regras, porém ainda não possui suporte adequado para a manipulação de regras. Isto se deve a que as regras são armazenadas no repositório de dados e faltam estruturas adequadas neste repositório para armazenar, independentemente, os elementos das regras. Estas estruturas incluem tabelas e mecanismos para auxiliar as operações de gerência sobre os elementos destas regras, como as meta-regras. Atualmente, na linguagem SQL3 e nos SGBDAs (comerciais), as informações sobre os elementos do *trigger*, a saber: evento, condição e ação são armazenados em uma única tabela, e os demais objetos do banco de dados, tais como tabelas e atributos, utilizados pelo *trigger*, são armazenados em tabelas diferentes.

Segundo a visão da linguagem SQL3, o dicionário de dados compõe-se de estruturas para armazenar dados e regras. O repositório de regras proposto, tem um conjunto de estruturas, especialmente criado, para o armazenamento das regras e considera que os dados são armazenados de forma separada a estas estruturas. Portanto, esta separação promove a independência das regras com relação aos dados, e da mesma forma que os dados dispõem de operações para sua manipulação e definição, também são atribuídas às regras operações para alteração e definição delas e de suas partes. Estes dois conjuntos, regras e dados, estão associados por meio dos objetos de banco de dados, utilizados pela regra, como tabelas, colunas e informações sobre usuários, entre outros. As operações para a alteração das regras são apresentadas a seguir.

5.2.1 Operações Sobre Elementos de uma Regra

O evento é um elemento criado em conjunto com a definição de uma regra. Desta forma, para cada evento no repositório de regras, deve haver pelo menos uma ocorrência de regra associada a ele. O evento é um elemento fortemente acoplado a regra. Portanto alteração, adição ou eliminação dele, influencia o restante dos elementos da regra. Este acoplamento se dá em função do seu tempo de execução, variáveis de transição e granularidade. Esta visão é sustentada pela linguagem SQL3. Um evento é um acontecimento que ocorre em um determinado momento. A eleição deste acontecimento, tais como inserção de dados, atualização de dados, conexão ao sistema, está associada às necessidades impostas pelo negócio ou pelo usuário, portanto existe uma semântica agregada a este evento, que se reflete ao longo dos demais elementos da regra.

Considerando que um dos objetivos deste trabalho é estender a Linguagem SQL3 no que tange às operações de gerência, são sugeridas algumas operações de gerência, especificamente para o elemento evento, tais como: alterar o evento de uma regra, adicionar um evento a uma regra, eliminar um evento de uma regra. Nem todas estas operações são aplicadas ao evento definido pelo usuário, por exemplo, a operação “alterar evento” aplicada sobre as regras do tipo **derivação** e **operacional**. A operação *Fire* é um mecanismo de execução de regras específico para este tipo de regra, derivação e operacional. Esta operação tem a finalidade de executar estes tipos de regras e, portanto, difere do evento de regras utilizado para disparar as regras **estímulo-resposta**.

As operações de gerência são mecanismos providos ao usuário para a gerência das

regras que estão armazenadas no repositório de regras. Sendo assim, é fortemente encorajador oferecer ao usuário um conjunto de mecanismos que não altere a complexidade ou adicione maior grau de complexidade do que ele está acostumado a lidar. Esta complexidade é inerente ao modelo de regras adotado pelos SGBDAs. Esta é uma das razões para estender a Linguagem SQL3, visto que é amplamente utilizada, tanto na indústria quanto no meio acadêmico, em detrimento a sugerir novas formas de operações de gerência de regras. O conjunto de operações de gerência sugerido leva em consideração esta necessidade, a saber: um conjunto de operações para a manipulação de regras e suas partes, que esteja em conformidade com a proposta da Linguagem SQL3 e que tenha aderência às sintaxes atualmente apresentadas pelo modelo de regras desta Linguagem.

5.2.1.1 Alterar o evento de uma regra

A semântica associada ao termo "alterar o evento" representa a mudança de alguma operação definida na especificação do evento, e esta alteração deve ser acompanhada do objeto que sofre a ação, por exemplo, mudar a sentença de "*INSERT ON Funcionário*" para "*UPDATE ON Funcionário*". Esta última sentença é composta da operação *UPDATE* e do objeto que sofre a ação, a tabela Funcionário. O mesmo procedimento, alterar, é válido para os eventos de sistema, meta dados e evento temporal. A alteração de um evento somente tem sentido para as regras do tipo estímulo/resposta, cujo evento pode assumir diversos tipos de operações. As regras do tipo derivação e operacionais possuem um único tipo de evento, definido como *Fire*.

A sintaxe definida para a modificação do evento de uma regra é composta de duas partes. A primeira parte da operação *ALTER RULE <nome-regra>* identifica a regra que terá o seu evento alterado. A segunda parte, *MODIFY EVENT TO <identificação da operação> > [{<OR identificação da operação> <objeto que sofre a ação>}] | <data_hora>* identifica o novo evento que substituirá o evento anteriormente definido na regra. Este evento pode ser simples ou composto. Da mesma forma que os demais eventos, o evento temporal (especificado pela data e hora), também pode ser modificado. A seguir, é apresentada a sintaxe da operação de modificação do evento de uma regra.

Sintaxe:

ALTER RULE <nome-regra>
MODIFY EVENT TO <identificação da operação> [{<OR identificação da operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;

<identificação da operação> ::= <evento dados> |
<evento sistema> |
<evento meta-dados> |
<identificação da data_hora> ::= <evento temporal>

< evento dados > ::= <INSERT> [{OR <UPDATE> | <DELETE> | <SELECT>}]
Objeto que sofre a ação ::= [OF {<lista de colunas> <virgula>} OR ON <nome tabela>]
< evento sistema > ::= <LOGON> [{OR <LOGOFF >}]
Objeto que sofre a ação ::= [OF <nome do usuário>]
< evento meta-dados > ::= <ALTER> [{OR <DROP> | <RENAME >}]
Objeto que sofre a ação ::= [TABLE <nome da tabela>]
< evento temporal > ::= <data_hora>

Os eventos dado, sistema e meta-dados, são compostos por um conjunto de operações, que representam os tipos de operações que podem ser utilizadas na sintaxe acima. Operações, específicas, de um tipo de evento não podem ser utilizadas em conjunto com operações de outros eventos, visto que, cada um deles tem associado uma semântica. O evento dado utiliza sentenças para a manipulação de dados, e estas sentenças auxiliam na manutenção das regras de negócio e na manutenção da integridade dos dados do banco de dados. O evento sistema está associado com acessos, por usuários, em nível de banco de dados, e o evento meta-dados, da mesma forma que o evento sistema, é usado para tarefas administrativas como auditoria e manutenção das tabelas do banco de dados. As operações definidas nos eventos, sistema e meta-dados, são utilizadas quando é necessário realizar tarefas de segurança, como por exemplo, evitar que modificações sejam realizadas no esquema de banco de dados ou registrar eventos ou mudanças que ocorreram neste esquema.

Na Figura 22 é ilustrado um exemplo de aplicação desta operação de modificação de

evento. Seja a regra "Atualiza_salario" e deseja-se modificar o evento "*UPDATE ON Funcionario*" para o evento "*INSERT ON Funcionario*"

```
CREATE RULE ATUALIZA_SALARIO
AFTER UPDATE ON Funcionario
FOR EACH ROW
DO
  BEGIN
  UPDATE Salario
  SET Sal_Func = Sal_Func + Sal_Func * :NEW.Indice
  WHERE Cod_Func = :NEW.Cod_Func;
END;
```

Figura 22 – Regra de negócio Atualiza_Salario

```
ALTER RULE ATUALIZA_SALARIO
MODIFY EVENT TO INSERT ON Funcionario;
```

A regra "Atualiza_salario" é modificada e assume o novo evento.

```
CREATE RULE ATUALIZA_SALARIO
AFTER INSERT ON Funcionario
FOR EACH ROW
DO
  BEGIN
    UPDATE Salario
    SET Sal_Func = Sal_Func + Sal_Func * :NEW.Indice
    WHERE Cod_Func = :NEW.Cod_Func;
  END;
```

Considerando o exemplo acima, tanto o evento *INSERT ON Funcionario* quanto o evento *UPDATE ON Funcionário* são compatíveis com o tipo de variáveis de transição

(*NEW*) utilizadas na ação da regra.

A alteração da regra, especificamente o evento <*UPDATE ON Funcionário*> pelo evento <*INSERT ON Funcionario*> gera uma mudança de estado no repositório de regras. Esta alteração implica que um novo registro é adicionado à tabela EVENTO e o evento anterior é eliminado desta tabela. Porém, se o evento <*UPDATE ON Funcionário*> está associado à outra regra, então a eliminação da referência entre este evento e a regra ATUALIZA_SALARIO ocorre somente na tabela REGRA_EVENTO. Na tabela EVENTO, o atributo <operação> assume a nova operação, *INSERT*, definida na sintaxe acima.

Se, ao invés de *INSERT ON Funcionário* fosse *SELECT OF <lista de atributos>*, a variável de transição não seria *NEW*, mas sim, *CURRENT*, adaptando-se ao contexto apresentado. Também, é importante considerar que o tempo de ativação não seria o mais indicado para a operação apresentada, pois para a operação *INSERT* utiliza-se o tempo de ativação *ON*. Estas inconsistências seriam observadas quando a regra fosse compilada. É função do usuário corrigir estas inconveniências, visto que um alerta seria gerado pelo sistema.

A abrangência de um *trigger* depende de seu evento. Quando o usuário especifica uma regra, geralmente o corpo da regra (condição e ação) está correlacionado com este evento. Portanto, substituir as operações de um evento, por outras operações pertencentes a outro tipo de evento, diferente, sem analisar e/ou redefinir o corpo da regra, pode gerar inconsistências. Por exemplo, um evento composto por mais de uma operação do tipo *UPDATE OR LOGON* não apresenta qualquer tipo de correlação entre elas, visto que são operações de domínios diferentes. Para evitar tais inconveniências, é elaborada uma meta-regra (Meta-Regra 7), que verifica as operações que são definidas na sintaxe de modificação de evento. No exemplo a seguir, é ilustrada esta inconsistência.

Seja a regra ATUALIZA_CARGO, cuja finalidade é atualizar o salário do funcionário, sempre que seu cargo é atualizado. O salário é aumentado em função de um índice específico.

A regra ATUALIZA_CARGO, apresentada na Figura 23a, possui um evento do tipo "*UPDATE ON* Cargo" e na ação desta regra são definidas variáveis de transição associadas a este evento. Ao modificar o evento para "*LOGON OF* Usuário*_1*", apresentada na Figura 23b, estas variáveis perdem o sentido. Isto ocorre porque a operação "*LOGON OF* Usuário*_1*" atua em nível de banco de dados, com a finalidade de acesso, pelo Usuário_1, ao banco de dados. Além da modificação do evento, é necessário ajustar a variável de transição, de *AFTER* para

ON. A granularidade da regra perde seu sentido, uma vez que o evento sistema não atua em nível de registros.

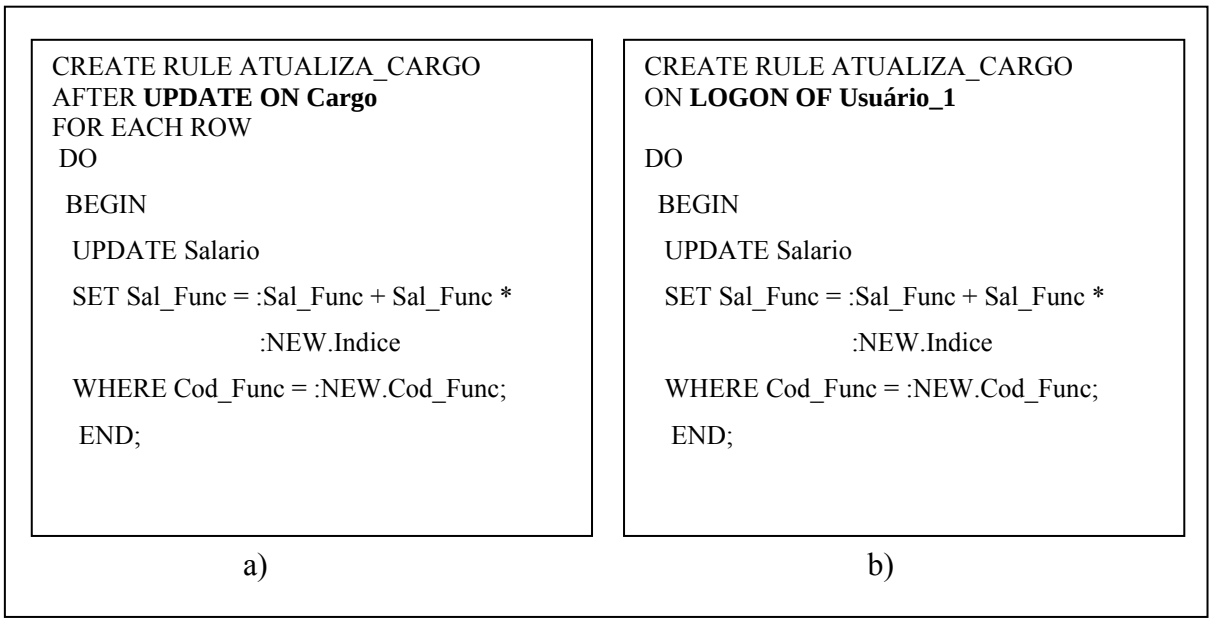

Figura 23 – Regra ATUALIZA_CARGO. Na alínea a, a regra possui um evento de banco de dados, e na alínea b, a mesma regra, teve o seu evento modificado por um evento sistema.

5.2.1.2 Eliminar o evento de uma regra

A semântica associada ao termo "eliminar o evento" , de uma regra, tem o sentido de duas ações a serem tomadas pelo SGBDA. A primeira ação é a própria eliminação do evento, e a segunda ação é a atribuição da operação *Fire* a regra. Portanto, a regra que tinha um perfil do tipo ECA, ECAA ou EA, característica das regras do tipo estímulo-resposta, para ter o perfil de uma regra do tipo CA, CAA ou A, características das regras do tipo operacional e derivação.

As regras estímulo-resposta podem ser definidas em função de diversos tipos de eventos, tais como: evento dados, evento meta-dados, evento sistema, e evento temporal. Todos estes eventos podem ser substituídos pela operação *Fire.*

Quando a operação para eliminação de um evento é aplicada sobre uma regra, o SGBDA executa os seguintes passos:

1) Elimina os registros da tabela REGRA_EVENTO que fazem referência à regra e aos eventos desta regra, e se não existe mais nenhuma ocorrência deste evento à outra regra, então se elimina a definição do evento da tabela EVENTO;

2) Adiciona um novo registro à tabela EVENTO, caso não haja nenhuma ocorrência deste evento nesta tabela, e uma ocorrência deste evento à tabela, na tabela REGRA_EVENTO;
3) Modifica-se as informações da regra, na tabela REGRA, de forma a adequar-se ao novo evento. Especificamente, os atributos <tipo-regra>, <granularidade>, <tempo de ativação> são alterados em função do novo evento;

A sintaxe para a eliminação de um evento, é a seguinte:

Sintaxe:

ALTER RULE <nome-regra>
DROP EVENT <identificação da operação> [{<OR identificação da operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;

<identificação da operação> ::= <evento dados> |
<evento sistema> |
<evento meta-dados> |
<identificação da data_hora> ::= <evento temporal>

< evento dados > ::= <INSERT> [{OR <UPDATE> | <DELETE> | <SELECT>}]
Objeto que sofre a ação ::= [OF {<lista de colunas> <virgula>} OR ON <nome tabela>]
< evento sistema > ::= <LOGON> [{OR <LOGOFF >}]
Objeto que sofre a ação ::= [OF <nome do usuário>]
< evento meta-dados > ::= <ALTER> [{OR <DROP> | <RENAME >}]
Objeto que sofre a ação ::= [TABLE <nome da tabela>]
< evento temporal > ::= <data_hora>;

No exemplo a seguir é apresentada uma regra com tempo de ativação *AFTER* e um evento dados com a operação *UPDATE* sobre o atributo "salário" da tabela EMP (empregado). A regra é do tipo ECA. A granularidade da regra é *FOR STATEMENT*, e a condição é uma consulta SQL, elementar, cuja finalidade é identificar se existe algum funcionário que recebe menos de 1500 reais, para os funcionários que foram admitidos até o

ano de 1990.

```
CREATE RULE SALARIO_FUNC
AFTER UPDATE OF salario ON EMP
FOR EACH STATEMENT
WHEN ((SELECT MIN(salario)
                FROM EMP
                WHERE data_admissao <= '01/12/1990') < 1500)
DO
                BEGIN
                        CALL PROCEDURE lista_empregado ('nomes');
                END;
```

ALTER RULE SALARIO_FUNC
DROP EVENT < UPDATE >< OF salario ON EMP **>;**

A regra "Atualiza_salario" é modificada e assume o novo evento.

```
CREATE RULE SALARIO_FUNC
WHEN ((SELECT MIN(salario)
                FROM EMP
                WHERE data_admissao < '01/01/1990') < 1500)
DO
                BEGIN
                        CALL PROCEDURE lista_empregado ('nomes');
                END;
```

No exemplo acima, o evento foi eliminado da especificação da regra, porém a operação *Fire* é responsável pelo disparo da regra SALARIO_FUNC. Com relação à condição e a ação, estas se mantêm sem alterações. É responsabilidade do usuário ajustar a regra de modo que estes dois elementos da regra não apresentem variáveis de transição, nem granularidade. Uma vez que a regra seja compilada, erros podem ser apresentados.

5.2.1.3 Adicionar um evento a uma regra

A semântica associada à operação de adição, de um evento, está associada às regras de derivação e operacional. Adicionar um evento implica eliminar o evento definido atualmente como *FIRE* e substituí-lo por um evento dados, evento sistema, meta-dados ou temporal. A evolução de uma regra do tipo derivação ou operacional para uma regra estímulo-resposta apresenta menor impedância que o seu processo inverso, isto porque não é necessário preocupar-se com os ajustes das variáveis de transição.

Com relação à granularidade da regra, é importante considerar que a condição da regra seja compatível com a granularidade definida para a regra. Por exemplo, a granularidade orientada a conjunto é utilizada quando a condição considerar consultas orientadas a conjunto de linhas. Por exemplo, existência de sentença em linguagem SQL, com funções de grupo como média, somas, entre outras.

O impacto de uma operação de adição de um evento no repositório de regras é descrito a seguir:

1) Eliminar os registros da tabela REGRA-EVENTO que fazem referência à regra e eliminar as ocorrências da tabela EVENTO, caso não existam ocorrências deste evento na tabela REGRA-EVENTO.
2) Adiciona um novo registro à tabela EVENTO, caso não existam ocorrências deste evento nesta tabela, e adequar os atributos <tipo-regra>, <granularidade>, <tempo de ativação>. Esta alteração implica na identificação da granularidade e do tempo de ativação da nova regra, que são definidos pelo usuário. O tipo da regra é identificado automaticamente.
3) Logo a seguir, adicionar as devidas ocorrências na tabela REGRA-EVENTO;

A sintaxe para a adição de um evento a uma regra é a seguinte:

Sintaxe:

ALTER RULE <nome-regra>
ADD EVENT <<identificação da operação> [{<OR identificação da
operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;
[ACTIVATION TIME <identificação do tempo de ativação>]
[GRANULARITY <identificação da granularidade>];

<identificação da operação> ::= <evento dados> |
<evento sistema> |
<evento meta-dados> |
<identificação da data_hora> ::= <evento temporal>

< evento dados > ::= <INSERT> [{OR <UPDATE> | <DELETE> | <SELECT>}]
Objeto que sofre a ação ::= [OF {<lista de colunas> <virgula>} OR ON <nome
tabela>]
< evento sistema > ::= <LOGON> [{OR <LOGOFF >}]
Objeto que sofre a ação ::= [OF <nome do usuário>]
< evento meta-dados > ::= <ALTER> [{OR <DROP> | <RENAME >}]
Objeto que sofre a ação ::= [TABLE <nome da tabela>]
< evento temporal > ::= <data_hora>;
identificação do tempo de ativação ::= BEFORE [{OR <AFTER> | <ON>}]
identificação da granularidade **::=** <FOR EACH ROW> [{OR <FOR STATEMENT>]

O tempo de ativação e a granularidade da regra são informações opcionais. Quando o usuário não especifica estes valores, o SGBDA assume os valores padrão. Para o tempo de ativação, o valor padrão é *BEFORE* e para a granularidade o valor padrão é *FOR STATEMENT*. Estes valores estão em conformidade com a linguagem SQL3.

A seguir é apresentada uma regra denominada CATEGORIA_CLIENTE que recebe alguns valores por meio de variáveis globais. A regra é do tipo “Ação”, cuja finalidade é identificar a categoria do cliente em função de seu salário.

```
VARIAVEIS GLOBAIS
(V_CodCli       Cliente.Cod_Cli%TYPE)
 V_TipVelho     Cliente.Tip_Cli%TYPE;
 V_TipNovo      Cliente.Tip_Cli%TYPE;
 V_SalCli       Cliente.Sal_Cli%TYPE;
CREATE RULE CATEGORIA_CLIENTE
DO
   BEGIN
              SELECT Sal_Cli, Tip_Cli       INTO V_SalCli, V_TipVelho
      FROM Cliente
      WHERE Cod_Cli = V_CodCli;
            IF V_SalCli >= 5000 THEN
                  V_TipNovo:= 'A';
                ELSIF V_SalCli < 5000 AND V_SalCli >= 3000 THEN
                  V_TipNovo:= 'B';
                ELSE
                  V_TipNovo:= 'C';
            END IF;
                IF V_TipNovo <> V_TipVelho THEN
                  UPDATE Cliente
                  SET Tip_Cli = V_TipNovo
                  WHERE Cod_Cli = V_CodCli;
            END IF;
  END;
```

ALTER RULE CATEGORIA_CLIENTE

ADD EVENT <UPDATE> <OF salario ON CLIENTE>

[ACTIVATION TIME <AFTER>**]**

[GRANULARITY <FOR EACH ROW>**];**

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
DO
    BEGIN
            SELECT Sal_Cli, Tip_Cli        INTO V_SalCli, V_TipVelho
        FROM Cliente
        WHERE Cod_Cli = V_CodCli;
            IF V_SalCli >= 5000 THEN
                    V_TipNovo:= 'A';
                ELSIF V_SalCli < 5000 AND V_SalCli >= 3000 THEN
                    V_TipNovo:= 'B';
                ELSE
                    V_TipNovo:= 'C';
            END IF;
                IF V_TipNovo <> V_TipVelho THEN
                    UPDATE Cliente
                    SET Tip_Cli = V_TipNovo
                    WHERE Cod_Cli = V_CodCli;
            END IF;
    END;
```

A regra CATEGORIA_CLIENTE que era do tipo operacional passou a ser do tipo evento-ação. O evento desta regra é uma operação de atualização sobre ao atributo salário da tabela cliente. Além disso, também foi atribuído o tempo de ativação *AFTER* e uma granularidade em nível de registro. Uma vez que a atribuição do novo evento foi definida na regra, nenhuma outra modificação necessita ser feita. O usuário tem a opção de ajustar a ação da regra e utilizar variáveis de transição, compatíveis com o evento que foi definido, por exemplo, ao invés de utilizar a variável V_CodCli, é possível utilizar *NEW*.CodCli.

5.2.1.4 Adicionar uma condição a uma regra

A semântica associada ao termo “adicionar uma condição” a uma regra, representa inserir uma condição a uma regra. As regras as quais esta operação pode ser aplicada são do tipo operacional e estímulo-resposta. A adição da condição implica em inserir um registro na tabela CONDIÇÃO do repositório de regras. A tabela CONDIÇÃO apresenta um identificador da condição, <id_condição> definido pelo próprio SGBDA e um identificador da regra, <id_regra>, a qual a condição pertence. O atributo <data_última_modificação> armazena a data da última operação realizada sobre a regra.

Para utilizar a operação de adição de uma condição a uma regra, é necessário que o usuário esteja a par de que a condição da regra está diretamente correlacionada com a granularidade da regra. A sintaxe de adição do elemento condição, é a seguinte:

Sintaxe:
ALTER RULE <nome-regra>
ADD CONDITION [<operador unário>] <elemento - condição>
[<conector> <elemento - condição>] **...**

<operador unário> ::= NOT
<conector> ::= AND | OR
< elemento - condição > ::= <condição - SQL3> | <predicado-variável-de-transição>

A primeira parte da sentença “ALTER RULE <nome-regra>” identifica a regra. A segunda parte “ADD CONDITION” identifica o tipo de expressão que pode ser adotada na condição. Nesta operação, é possível adotar o operador NOT.

Para controlar a aplicação desta operação de gerência sobre uma regra, é necessário atribuir uma meta-regra, conforme é apresentado na Tabela 5.1

Tabela 5.1 –Meta-regra relacionada à adição de uma condição de uma regra..

Regra	Descrição
Meta-Regra 8	Somente as regras do tipo EA, A podem receber o elemento condição.

A regra anterior CATEGORIA_CLIENTE não possui uma cláusula condição, visto que é uma regra do tipo EA. A esta regra será atribuída uma condição por meio da operação apresentada acima. Primeiro é apresentada a sintaxe e depois o resultado desta atribuição.

```
ALTER RULE < CATEGORIA_CLIENTE>
ADD CONDITION (( SELECT          saldo_medio
                                 FROM  cliente
                                 WHERE          cod_cli = NEW.Codcli ) < 10000,00 )
```

Esta condição retorna “verdadeiro” se existem registros cujo salário do cliente é maior que mil reais. Considerando a condição verdadeira, então o cliente pode assumir uma categoria (A, B ou C). Clientes sem categoria não podem solicitar empréstimos ao banco. Esta regra de negócio é valida para todos os clientes. A regra CATEGORIA_CLIENTE é apresentada a seguir:

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
WHEN (( SELECT      salario
                    FROM          cliente
                    WHERE         cod_cli = NEW.Codcli ) > 1000,00 )
DO
    BEGIN
                A ação é a mesma definida anteriormente na regra …
      END;
```

Na condição desta regra, é adicionada uma variável de transição que se adequa ao evento definido previamente na regra. É responsabilidade do usuário garantir que estas variáveis sejam aplicadas de forma adequada na condição. Da mesma forma, é importante garantir que a condição seja adequada para o emprego da granularidade definida na regra.

5.2.1.5 Eliminar a condição de uma regra

Uma regra pode ser alterada em função da eliminação de sua condição. A eliminação de uma condição é possível para os tipos de regras estímulo-resposta que sejam definidas com uma única ação e para os tipos de regras derivação que possuam também uma única ação. Quando estes tipos de regras possuem ações primária e secundária, então a eliminação da condição não é permitida pelo SGBDA, visto que não existe uma regra somente com duas ações, sem o elemento condição. Para controlar a aplicação desta operação de gerência sobre uma regra, é necessário atribuir uma meta-regra, conforme é apresentado na Tabela 5.2.

Tabela 5.2 – Meta-regra relacionada à eliminação de condição de uma regra.

Regra	Descrição
Meta-Regra 9	Somente as regras do tipo ECA, CA podem ser alteradas para regras do tipo EA e A.

A sintaxe para eliminação da condição de uma regra é a seguinte:

Sintaxe:
ALTER RULE <nome-regra>
DROP CONDITION;

O efeito desta regra sobre o repositório de regras é a eliminação de um registro da tabela CONDIÇÃO. Não é necessário fazer referência a que condição está sendo eliminada, porque ela é única e pode ser identificada por meio do nome da regra.

Seja a regra a seguir utilizada para guardar o histórico dos salários dos funcionários de uma empresa. A informação sobre os salários é mantida em um histórico, toda vez que um salário é atualizado. Além disso, o usuário é avisado de que a média salarial é maior que R$ 10.000, 00.

Exemplo:

```
CREATE RULE  SALARIO_FUNC
AFTER UPDATE OF salario ON FUNCIONARIO
WHEN ((Select AVG(salario)
                FROM FUNCIONARIO) > 10.000)
DO
        BEGIN
                INSERT INTO HISTORICO_SALARIO_FUNCIONARIO
              ( SELECT * FROM OLD_TABLE);
              SIGNAL (- 500, “A media salarial dos funcionários não pode ser maior que
                        R$ 10.000,00”).
        END;
```

ALTER RULE < SALARIO_FUNC >
DROP CONDITION;

```
CREATE RULE  SALARIO_FUNC
AFTER UPDATE OF salario ON FUNCIONARIO
DO
        BEGIN
                INSERT INTO HISTORICO_SALARIO_FUNCIONARIO
              ( SELECT * FROM OLD_TABLE);
              SIGNAL (- 500, “A media salarial dos funcionários não pode ser maior que
                        R$ 10.000,00”).
        END;
```

5.2.1.6 Alterar a condição de uma regra

As regras que podem ter sua condição alterada são regras do tipo estímulo-resposta e derivação. Alterar uma condição implica em modificar um registro na tabela CONDIÇÃO do repositório de regras: Isto implica em modificar o atributo “definição_condição” e atualizar o atributo “data_última_modificação”. Para aplicar esta operação de gerência, é necessário que a regra seja do tipo estímulo-resposta e derivação. Uma meta-regra deve ser utilizada para garantir a correta aplicação da operação sobre um tipo de regra previamente definido,

conforme apresentado na Tabela 5.3:

Tabela 5.3 – Meta-regra relacionada à alteração da condição de uma regra

Regra	Descrição
Meta-Regra 10	Somente regras do tipo ECA, ECAA, CA, CAA podem ter sua condição alterada.

A sintaxe de alteração é apresentada a seguir:

ALTER RULE <nome-regra>
MODIFY CONDITION TO [<operador unário>] <elemento - condição>
[<conector> <elemento - condição>] **...**

<operador unário> ::= NOT
<conector> ::= AND | OR
< elemento - condição > ::= <condição - SQL3> | <predicado-variável-de-transição>

Seja a seguinte regra:

```
CREATE RULE SALARIO_FUNC
AFTER UPDATE OF salario ON EMP
FOR EACH STATEMENT
WHEN ((SELECT MIN(salario)
                FROM EMP
                WHERE data_admissao <= ’01/12/1990’) < 1500)
DO
              BEGIN
                    CALL PROCEDURE lista_empregado (‘nomes’);
              END;
```

Esta regra tem a finalidade de identificar os funcionários que foram admitidos antes de dezembro de 1990 e que recebem um salário menor que R$ 1500,00.

Considerando que na empresa não trabalham mais os funcionários que foram admitidos antes da data de dezembro de 1990, esta regra perdeu o sentido para a empresa. O

usuário voltou-se para uma lei que prevê que os funcionários que trabalham pelo menos 6 horas diárias não podem receber menos de um salário mínimo. Assim, aplica-se a operação de gerência para modificação da condição conforme especificado abaixo:

ALTER RULE < SALARIO_FUNC >
MODIFY CONDITION TO ((SELECT MIN(salario)
FROM EMP) < (350)

```
CREATE RULE SALARIO_FUNC
AFTER UPDATE OF salario ON EMP
WHEN ((SELECT MIN(salario)
        FROM EMP ) < (350)
DO
        BEGIN
            CALL PROCEDURE lista_empregado ('nomes');
        END;
```

Para aplicar esta operação, é necessário que o usuário considere a granularidade da regra e o tempo de ativação da mesma.

5.2.1.7 Adicionar uma ação a uma regra

Uma regra deve apresentar pelo menos uma ação. A adição de uma ação a uma regra representa a adição de uma ação secundária. Para que seja possível atribuir mais uma ação, a regra deve possuir uma condição associada a ela. O reflexo no repositório de regras, em função desta adição, é um registro associado a uma regra. O SGBDA atribui um código para esta ação, adicionando um valor ao atributo <id_ação> e atribui o valor secundário ao atributo <categoria>. A sintaxe para a adição de uma ação a uma regra é a seguinte:

Sintaxe:

ALTER RULE <nome_regra>
ADD SECONDARY ACTION <ação secundária>

<ação secundária> ::= <sentença procedimental SQL> |

CALL <nome procedimento> |
<bloco SQL>
<operação -regra> <nome -regra>

<bloco SQL> ::= BEGIN
{<sentença procedimental SQL> <ponto e vírgula>} ...
END;

A primeira parte desta sintaxe “ALTER RULE <nome_regra>” identifica a regra que será alterada. A segunda parte da operação de gerência, ADD SECONDARY ACTION identifica a ação a ser inserida na regra. Esta ação é composta de uma sentença denominada “ação secundária”. Esta sentença pode ser uma operação válida dentro da linguagem procedimental SQL3. Quando um conjunto de sentenças deve ser especificado, então se utiliza um bloco SQL, <bloco SQL>, ou uma chamada a procedimento, <CALL nome procedimento>. Além disso, é permitido definir uma operação - regra, <operação-regra>, definida pelas operações habilitar, desabilitar ou disparar uma regra, por meio de um evento *Fire*. Para garantir a consistência do repositório de regras, é necessário que algumas meta-regras atuem quando o usuário utilizar esta operação de gerência. Estas regras são apresentadas na Tabela 5.5:

É apresentado um exemplo de aplicação desta operação de gerência. Considere a regra CATEGORIA CLIENTE, apresentada anteriormente. Quando a condição for verdadeira, a ação é executada. Caso contrário, nada acontece. Porém, o gerente pode ser notificado de que o usuário não foi aprovado, no que se refere ao seu saldo médio, para possíveis empréstimos. Ao adicionar uma ação secundária, é possível incorporar esta informação, necessária para o gerente.

ALTER RULE < CATEGORIA_CLIENTE >
ADD SECONDARY ACTION <CALL PROCEDURE Saldo_Médio (NEW.Codcli)>;

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
WHEN (( SELECT      saldo_medio
                FROM        cliente
                WHERE       cod_cli = NEW.Codcli ) < 1000,00 )
DO
    BEGIN
                A ação é a mesma definida anteriormente na regra …
      END;
ELSEDO
    BEGIN
                CALL PROCEDURE Saldo_Médio (NEW.Codcli)
      END;
```

O usuário tem a responsabilidade de evitar que as variáveis de transição, na ação secundária, não sejam compatíveis com o evento da regra.

Uma vez que a operação é executada, o SGBDA atualiza a informação referente ao registro da regra afetada na tabela REGRA. O atributo <tipo_regra> é atualizado e deixa de assumir o valor ECA para assumir o valor ECAA. Esta atualização ocorre em função de uma meta-regra, conforme a Tabela 5.4.

Tabela 5.4 – Meta-regras relacionadas à alteração da ação de uma regra

Regra	Descrição
Meta-Regra 11	Somente é permitido adicionar uma ação a regras do tipo ECA e CA
Meta-Regra 12	As regras do tipo ECAA, CAA não podem receber outra ocorrência de ação

5.2.1.8 Modificar a ação de uma regra

A operação de gerência para modificar a ação de uma regra pode ser utilizada em todos os tipos de regras, derivação, operacional, e estímulo-resposta. No repositório de regras, modificar a ação implica em eliminar o registro da regra em questão e adicionar um novo registro, na tabela AÇÃO.

O usuário, ao atribuir uma nova ação, deve considerar o tipo de evento da regra. Desta forma eliminam-se possíveis problemas com variáveis de transição que não são compatíveis com este evento. Da mesma forma, em uma ação, é possível que existam outras regras, disparadas pela operação *Fire*. O SGBDA ao compilar a ação, verifica a existência desta composição de regras e armazena na tabela REGRA-COMPOSIÇÃO esta nova seqüência especificada. Quando forem definidas operações de banco de dados na ação, é verifica se existem outras regras que podem ser disparadas em função destas operações, esta informação é armazenada na tabela AÇÃO_EV.

A sintaxe para modificação da ação é.
Sintaxe:

ALTER RULE <nome_regra>
MODIFY [PRIMARY | SECONDARY] ACTION <ação primaria | secundária >;

<ação primaria | secundária> ::= <sentença procedimental SQL> |
CALL <nome procedimento> |
<bloco SQL>
<operação -regra>

<bloco SQL> ::= BEGIN
{<sentença procedimental SQL> <ponto e vírgula>} ...
END;

Seja a regra Publicacao_Prof apresentada a seguir: Um usuário deseja saber a titulação de um professor, com a finalidade de identificar se ele atingiu a quantidade de publicações

(somente artigos) para o ano corrente. Esta regra pode ser disparada toda vez que ele, o professor, entra em férias. A regra pode ser alterada para que o usuário possa verificar se o professor possui também publicações em livros.

```
CREATE RULE Publicacao_Prof
ON SELECT OF titulacao ON PROFESSOR
FOR EACH ROW
WHEN (current.status = 'Ferias')
DO
        BEGIN
                SELECT COUNT (publicacao) INTO V_quantidade
                FROM        PROFESSOR
                WHERE       codProf = CURRENT.codProf
                AND         ano = current_year;

                CALL PROCEDURE nro_publicacao (codProf, V_quantidade)
        END;
```

ALTER RULE < Publicacao_Prof >

MODIFY [PRIMARY] ACTION

```
< BEGIN
                SELECT COUNT (publicacao) INTO V_quantidade
                FROM        PROFESSOR
                WHERE       codProf = CURRENT.codProf
                AND         ano = current_year;
                SELECT COUNT (livros) INTO V_livros
                FROM        LIVROS
                WHERE       codProf = CURRENT.codProf
                AND         ano = current_year;
                CALL PROCEDURE nro_publicacao (codProf, V_quantidade)
                CALL PROCEDURE livro_publicacao (codProf, V_livros)
END;
>;
```

A nova ação inclui uma consulta SQL e uma nova chamada a procedimentos.

Portanto, após a aplicação desta operação de modificação do elemento ação, a regra fica da seguinte maneira:

```
CREATE RULE Publicacao_Prof
ON SELECT OF titulacao ON PROFESSOR
FOR EACH ROW
WHEN (current.status = 'Ferias')
DO
BEGIN
        SELECT COUNT (publicacao) INTO V_quantidade
        FROM    PROFESSOR
        WHERE   codProf = CURRENT.codProf
        AND     ano = current_year;
        SELECT COUNT (livros) INTO V_livros
        FROM    LIVROS
        WHERE   codProf = CURRENT.codProf
        AND     ano = current_year;
        CALL PROCEDURE nro_publicacao (codProf, V_quantidade)
        CALL PROCEDURE livro_publicacao (codProf, V_livros)
END;
```

O usuário deve garantir que a nova ação deve se adaptar ao novo contexto apresentado, em função do evento da regra. Para a nova ação, foi adicionada uma nova consulta SQL com a mesma variável de transição e uma chamada a procedimentos.

5.2.1.9 Eliminar a ação de uma regra

A eliminação de uma ação pode ocorrer somente em regras do tipo estímulo-resposta e derivação, sempre que as regras forem portadoras de uma ação primária e uma ação secundária. Eliminar implica em excluir um registro da tabela AÇÃO. Também devem ser eliminados registros, quando houver alguma ocorrência que envolva esta regra, da tabela AÇÃO_TAB e AÇÃO_EV. É possível eliminar uma ação primária ou uma ação secundária. Em ambos os casos, o reflexo desta modificação no repositório de regras são as eliminações dos registros referentes as tabelas mencionadas anteriormente. Na tabela REGRA, o atributo <tipo_regra> é modificado para assumir o novo valor. Para manter a consistência do repositório de regras, é necessário que o SGBDA realize os devidos ajustes, sobre os dados da

ação resultante. Por exemplo, ao eliminar uma ação primária, a regra fica com uma ação secundária. Uma regra não pode ter uma ação secundária sem a presença de uma ação primária. Assim, ao eliminar uma ação primária, a ação secundária assume, automaticamente, o papel de ação primária. O atributo <categoria>, na tabela AÇÃO, muda o valor armazenado de secundário para primário. A seguir apresenta-se na Tabela 5.5, a regra de consistência relacionada com a eliminação de uma ação de uma regra.

Tabela 5.5 – Meta-regra relacionada à eliminação da ação de uma regra

Regra	Descrição
Meta-Regra 13	Somente as regras do tipo ECAA e CAA podem eliminar a ação da regra.

A seguir é apresentada a sintaxe da operação de gerência para a eliminação da ação de uma regra.

ALTER RULE <nome_regra>
DROP [PRIMARY | SECONDARY] ACTION;

Seja a regra CATEGORIA_CLIENTE definida a seguir, com duas ações, primária e secundária. Ao aplicar a operação acima, a regra deve resultar somente em uma ação.

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
WHEN (( SELECT      saldo_medio
                FROM        cliente
                WHERE       cod_cli = NEW.Codcli ) < 1000,00 )
DO
    BEGIN
        A ação é a mesma definida anteriormente na regra CATEGORIA_CLIENTE …
      END;
ELSEDO
                CALL PROCEDURE Saldo_Médio (NEW.Codcli)
   END;
```

ALTER RULE < CATEGORIA_CLIENTE >
DROP [PRIMARY] ACTION;

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
WHEN (( SELECT      saldo_medio
                FROM        cliente
                WHERE       cod_cli = NEW.Codcli ) < 1000,00 )
DO
    BEGIN
        CALL PROCEDURE Saldo_Médio (NEW.Codcli)
    END;
```

Ao executar esta regra, verifica-se o valor resultante da avaliação da condição, caso seja verdadeira, a ação é executada, caso contrário, nada acontece. O SGBDA conhece o tipo de execução que deve ser realizada, devido à informação armazenada no atributo <tipo_regra>. Portanto é fundamental que após a eliminação da ação, este atributo deve ser ajustado a sua nova configuração.

A regra passa a ter uma nova semântica, após a eliminação de sua ação primária. É responsabilidade do usuário garantir que a regra resultante satisfaça os requisitos no negócio.

5.2.1.10 Alternar as ações de uma regra

A operação de gerência "alternar as ações de uma regra" tem a finalidade de trocar a categoria, de primária para secundária e vice-versa, de uma ação. Esta operação é aplicada em regras estímulo-resposta e derivação, que possuem duas ações.

O impacto gerado por esta operação no repositório de regras ocorre somente na tabela AÇÃO. O atributo <categoria> inverte os valores, de primária para secundária e vice-versa. Esta operação tem a finalidade de evitar que o usuário elimine uma ação, segundo o processo explicado na operação de eliminação de uma ação, e depois volte a definí-la.

A sintaxe desta operação é a seguinte:

ALTER RULE < nome_regra >
CHANGE ACTION FROM [PRIMARY] TO [SECONDARY];

A primeira parte da sintaxe identifica a regra e a segunda parte tem a finalidade de alternar a ação primária para que assuma o papel de ação secundária e a ação secundária assuma o papel de ação primária.

Considerando a regra CATEGORIA_CLIENTE, apresentada anteriormente, após a aplicação da regra acima, o resultado é apresentado a seguir:

ALTER RULE < CATEGORIA_CLIENTE >
CHANGE ACTION FROM [PRIMARY] TO [SECONDARY];

```
CREATE RULE CATEGORIA_CLIENTE
AFTER UPDATE OF salario ON CLIENTE
FOR EACH ROW
WHEN (( SELECT      saldo_medio
        FROM        cliente
        WHERE       cod_cli = NEW.Codcli ) < 1000,00 )
DO
            CALL PROCEDURE Saldo_Médio (NEW.Codcli)
ELSEDO
  BEGIN
       A ação é a mesma definida anteriormente na regra CATEGORIA_CLIENTE
   END;
```

5.2.2 Operações Sobre Regras

As operações relacionadas às propriedades de uma regra dizem respeito à regra como um todo, tratando a regra de forma uníssona. Estas operações podem habilitar ou desabilitar uma regra, ou um conjunto de regras, agrupar regras e eliminar este agrupamento.

As operações de gerência foram definidas em dois grupos, um grupo que trata da gerência de seus elementos e outro grupo que trata da gerência das propriedades da regra.

5.2.2.1. Habilitar ou Desabilitar uma regra

Uma regra pode ser habilitada ou desabilitada por uma operação de gerência ou, automaticamente, por meio de outra regra. Toda regra quando é especificada no repositório de regras assume o valor padrão “Habilitada”. Para habilitar ou desabilitar uma regra usa-se a seguinte operação de gerência.

ENABLE [OR DISABLE] RULE < nome_regra >;

O impacto desta operação no repositório de regras é a alteração do valor do atributo <status> na tabela REGRA. Quando <status> possui o valor “H”, a regra está habilitada, e quando este atributo possui o valor “D”, a regra está desabilitada.

Supondo que a regra CATEGORIA_CLIENTE esteja habilitada, no repositório de regras. Para desabilitá-la basta aplicar a seguinte operação.

DISABLE RULE < CATEGORIA_CLIENTE>;

Uma vez que a operação sucedeu de forma satisfatória, a regra passa a ser desabilitada.

5.2.2.2 Agrupamento de regras

Um conjunto de regras pode ser agrupado com a finalidade de realizar operações de gerência sobre este grupo. Ao invés de aplicar, repetidamente, a mesma operação para cada regra, é possível fazê-lo uma única vez por meio da definição de grupo de regras.

A sintaxe de agrupamento de regras é a seguinte:

CREATE RULESET < nome_grupo_regra >
ADD RULE [lista de regras**]**
[OR DELETE RULE [lista de regras**]]**

A primeira parte desta sintaxe identifica o nome do grupo que constitui o conjunto de regras. A segunda parte tem a finalidade de adicionar regras a este grupo e a terceira parte a finalidade de eliminar regras do grupo de regras. A seguir é apresentado um agrupamento de regras.

CREATE RULESET < universidade >
ADD RULE Publicacao_Prof; Salário_Func

O conjunto de regras denominado “Universidade” contém as duas regras Publicacao_Prof e Salário_Func.

5.2.2.3 Eliminar Grupo de Regras

A operação de gerência utilizada para eliminar um agrupamento de regras apresenta a funcionalidade oposta da operação *CREATE RULESET*. Portanto, para eliminar um agrupamento de regras é necessário que elas estejam agrupadas. A sentença utilizada para eliminação de um grupo de regas é a seguinte:

DROP RULESET <nome_grupo_regra>;

Para eliminar o agrupamento de regras "universidade", utiliza-se a seguinte operação:

DROP RULESET universidade ;

5.2.2.4 Eliminar uma regra

A eliminação de uma regra do repositório de regras representa a exclusão de todos os elementos associados à regra, destes elementos com os objetos do banco de dados e às informações da regra. Na tabela CONDIÇÃO, são eliminadas todas as condições que possuem referência à regra a ser eliminada e as informações contidas na tabela CONDIÇÃO_TAB. Este procedimento é realizado também na tabela AÇÃO e suas informações contidas nas tabelas AÇÃO_TAB e AÇÃO_EV. Com relação à tabela EVENTO, estes também devem ser eliminados, assim como as informações das tabelas REGRA_EV, EV_COL.

Quando uma regra a ser eliminada é composta de outras regras, então não existem restrições para esta eliminação, e o registro referente à regra a ser eliminada é excluído da tabela REGRA_COMPOSIÇÃO. O processo inverso deve ser reavaliado, visto que a regra a ser eliminada pode fazer parte de outra regra, neste caso, a eliminação não é trivial, pelo fato de que estas regras fazem parte de um processo de negócio, e necessita da intervenção do usuário. O usuário deve estar ciente de que existe uma dependência entre a regra a ser eliminada e a regra que é composta de várias regras. Para manter a consistência do repositório de regras, é necessário avisar o usuário da existência desta dependência e solicitar a sua aprovação, no que se refere à eliminação da regra. Esta aprovação pode ser realizada por meio de uma autorização ou notificação: por exemplo:"- A regra a ser eliminada faz parte de outra regra. Deseja seguir com o procedimento de eliminação da regra?" ou "A regra a ser

eliminada faz parte de outra regra.”.

A sintaxe da operação de gerência para a eliminação de uma regra é:

DROP RULE <nome_regra>;

Para eliminar a regra CATEGORIA_CLIENTE utiliza-se a sentença acima:

DROP RULE CATEGORIA_CLIENTE;

5.2.2.5 Habilitar ou Desabilitar um Conjunto de Regras

Habilitar e Desabilitar um conjunto de regras é possível somente quando as regras pertençam a um determinado grupo. O efeito desta operação no repositório de regras é o mesmo que para uma única operação, habilitar ou desabilitar regras. A sintaxe da operação de gerência para habilitar ou desabilitar um conjunto de regras é a seguinte:

ENABLE [OR DISABLE] RULESET< nome_grupo_regra >;

Para habilitar um conjunto de regras, é necessário aplicar a operação acima, conforme apresentado no exemplo a seguir, para o agrupamento de regras denominado “universidade”.

ENABLE RULESET universidade;

5.3 NAVEGADOR PARA A GERÊNCIA DE REGRAS

Vários esforços têm sido realizados, e apresentados na literatura, para oferecer ao usuário um ambiente no qual seja possível realizar manutenções em regras e ao mesmo tempo visualizar o comportamento e a interatividade entre elas (CHAKRAVARTHY; TAMIZUDDIN; ZHOU, 1995; FORS, 1995; LEMKE; MANTHEY, 1997; MENS, 1998; BERNDTSSON; MELLIN; HÖGBERG, 1999; DIAZ; JAIME, 2000; DITTRICH, et al., 2000). Estes esforços, traduzidos por meio de ferramentas, ocorre em função da natureza das

regras, estímulo-resposta, derivação e operacionais, que têm a característica de dispararem outras regras, implicitamente ou explicitamente. Tratando-se de regras estímulo-resposta, o encadeamento acontece de forma implícita, o que torna difícil a sua visualização. Esta situação torna-se mais complexa quando os projetistas das regras não são os mesmo que desenvolveram a aplicação. As ferramentas existentes consideram um modelo de regras mais simples, apenas os tipos ECA e EA de regras, sugerido pela linguagem SQL3. Como conseqüência, um modelo significativamente mais rico em semântica, exige novos esforços para a visualização destes encadeamentos.

Foi proposto um repositório de regras que suporte estes novos tipos de regras, e neste repositório foram elaboradas estruturas que armazenam informações sobre o encadeamento de regras, seja este encadeamento disparado implicitamente quando explicitamente.

Neste trabalho, é apresentado um protótipo de uma ferramenta para auxiliar o usuário, a consultar informações sobre as regras, contidas no repositório de regras, os elementos destas regras e seus relacionamentos. De posse destas informações, é possível tomar decisão sobre os processos de negócio especificados por meio destas regras. Este protótipo se diferencia dos demais porque tem, a seu favor, um repositório de regras que trata o armazenamento de regras, de suas partes, e de seus relacionamentos.

Na Figura 24 apresenta-se a barra de tarefas do protótipo de navegação de regras. O protótipo possui algumas palavras reservadas (Arquivo, Editar, Operações, Relatórios e Ajuda) e figuras de atalho. O "Arquivo" tem a finalidade gravar, importar, exportar, e ver as propriedades das regras. "Editar" tem a finalidade de editar um texto, desfazer operação realizada, refazer esta operação. "Operações" possui todas as operações de gerência de regras possíveis de serem realizadas sobre o repositório de regras, inclusive solicitar informações sobre o encadeamento de regras, a partir de uma regra. "Relatórios" possui alguns padrões de relatórios e "Ajuda" possui informações de como utilizar o navegador de regras.

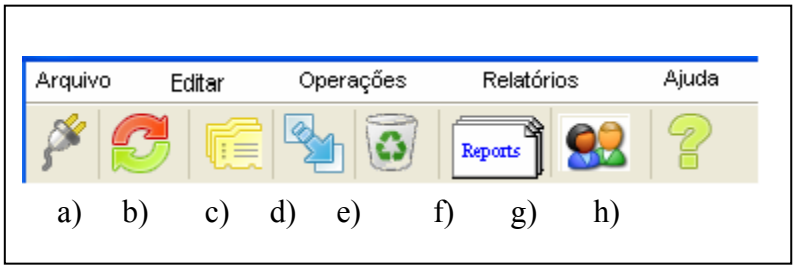

Figura 24 – Barra de tarefas do protótipo

Na Figura 24 alínea a, apresenta-se um atalho, cuja finalidade é estabelecer conexão a um determinado repositório de regras. Na alínea b, as duas setas representam uma ação para atualizar a tela do aplicativo (*refresh*). Na alínea c, o conjunto de folhinhas amarelas, tem a finalidade de apresentar as características da regra selecionada. Na alínea d, a seta em azul representa o encadeamento de regras, a partir de uma regra assinalada. A cesta de lixo, na alínea e, é uma metáfora indicando as regras, grupos ou partes destas que foram eliminadas pelas operações responsáveis por eliminar uma regra (seção 5.2.2.4), um grupo de regras (seção 5.2.2.3), eliminar um evento (seção 5.2.1.2), condição (seção 5.2.1.5) ou ação (seção 5.2.1.9). O conjunto de folhas representa os relatórios que podem ser gerados, conforme a necessidade do usuário. Na alínea g, é ilustrado um símbolo de agrupação de regras, com a finalidade de apresentar os grupos de regras existentes. O sinal de interrogação, na alínea h, serve para chamar a ajuda *on-line*, e buscas por palavra-chave.

Na Figura 25 é ilustrado o protótipo de um navegador de regras. Uma vez conectado ao SGBDA, é possível obter uma imagem do repositório de regras.

Neste repositório são definidos dois grupos de regras, definidos como alocação de recursos e empréstimos. Estes grupos são identificados pela seta seguida do número 1.

Neste navegador, são apresentadas as informações sobre os eventos, indicados pela seta de número 2, e os tipos de regras armazenadas no repositório, indicado pela seta de número 3. Estas regras fazem parte do grupo de regras "RegrasEmpréstimo" que possui diversos tipos de regras (ECA, ECAA, EA, CAA, CA, A). Para cada tipo de regra, são identificados seus respectivos nomes, conforme apresentado pela seta 4. As informações referentes ao encadeamento implícito de regras são identificadas pela seta 5. Neste item, são apresentadas as regras responsáveis pelo disparo de outra regra e a regras a serem disparadas, como também o evento responsável pelo encadeamento. A composição de regras é identificada pela seta 6. Neste item, são ilustradas as regras que fazem parte de uma composição de regras, e sua conseqüente prioridade (por exemplo, Nome_Regra(1)). A seta 7 apresenta as permissões que foram dadas para o grupo de regras, da mesma forma, apresenta-se o responsável pelo grupo de regra gerado. A sete 8 indica as propriedades da regra.

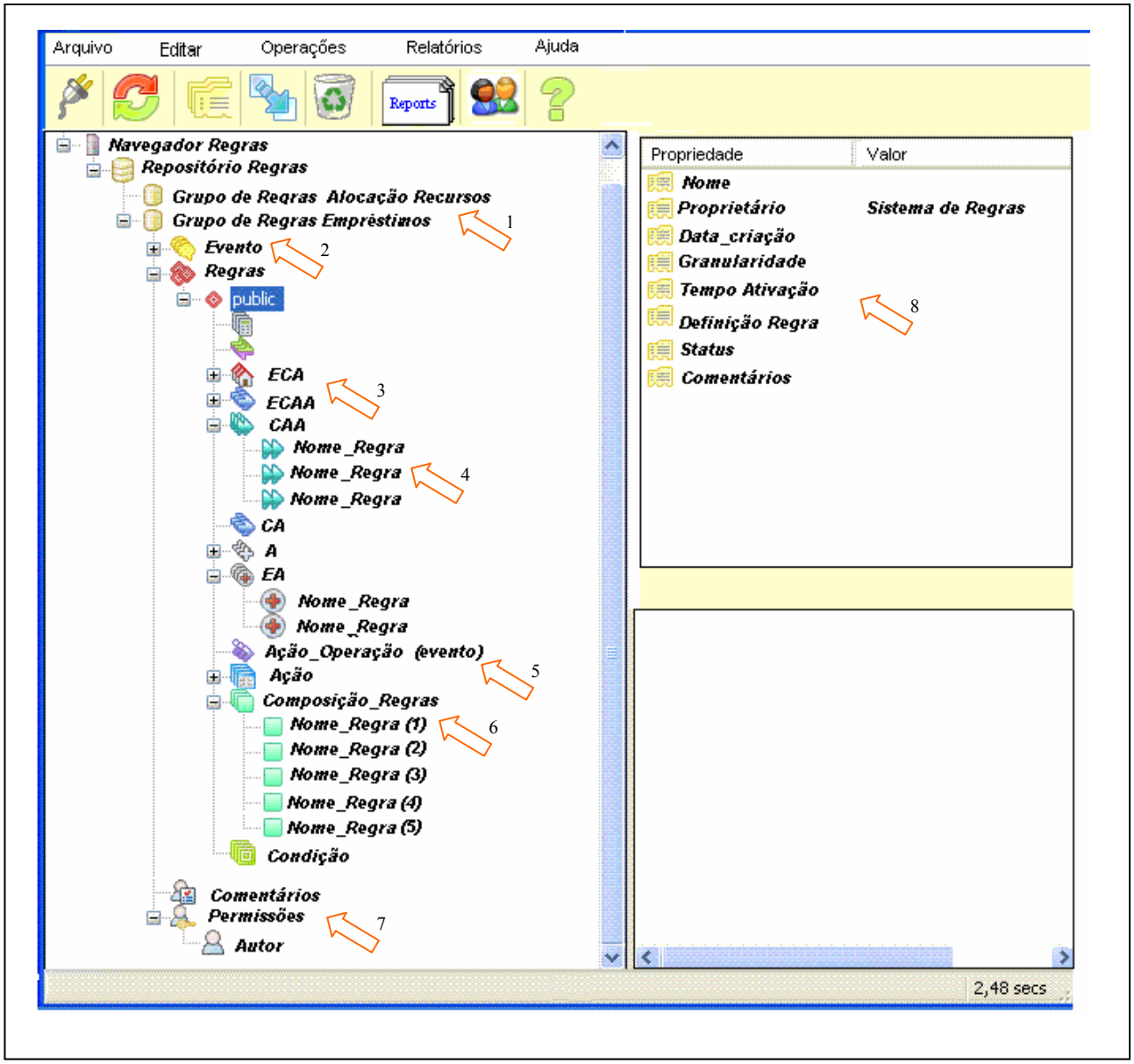

Figura 25 - Protótipo de um navegador de regras para o repositório de regras

5.4 REALIZAÇÃO DA PROPOSTA EM UM SBDA GENÉRICO

As funcionalidades de um SGBDA são representadas um conjunto de capacidades ativas tais como, detecção de eventos, execução de regras, compilação de regras, plano de execução de regras, entre outros. Estas capacidades podem ser identificadas por meio de uma arquitetura, de um SGBDA, conforme ilustrado na Figura 26. Uma arquitetura de um SGBDA determina a funcionalidade e os componentes necessários para garantir a realização delas

(PATON, 1998).

Este trabalho propõe uma linguagem de Gerência de Regras, como extensão da linguagem de regras adotada neste trabalho e descrita no capítulo 3. Para adequar a arquitetura do SGBDA a esta proposta, é necessário estender alguns de seus componentes.

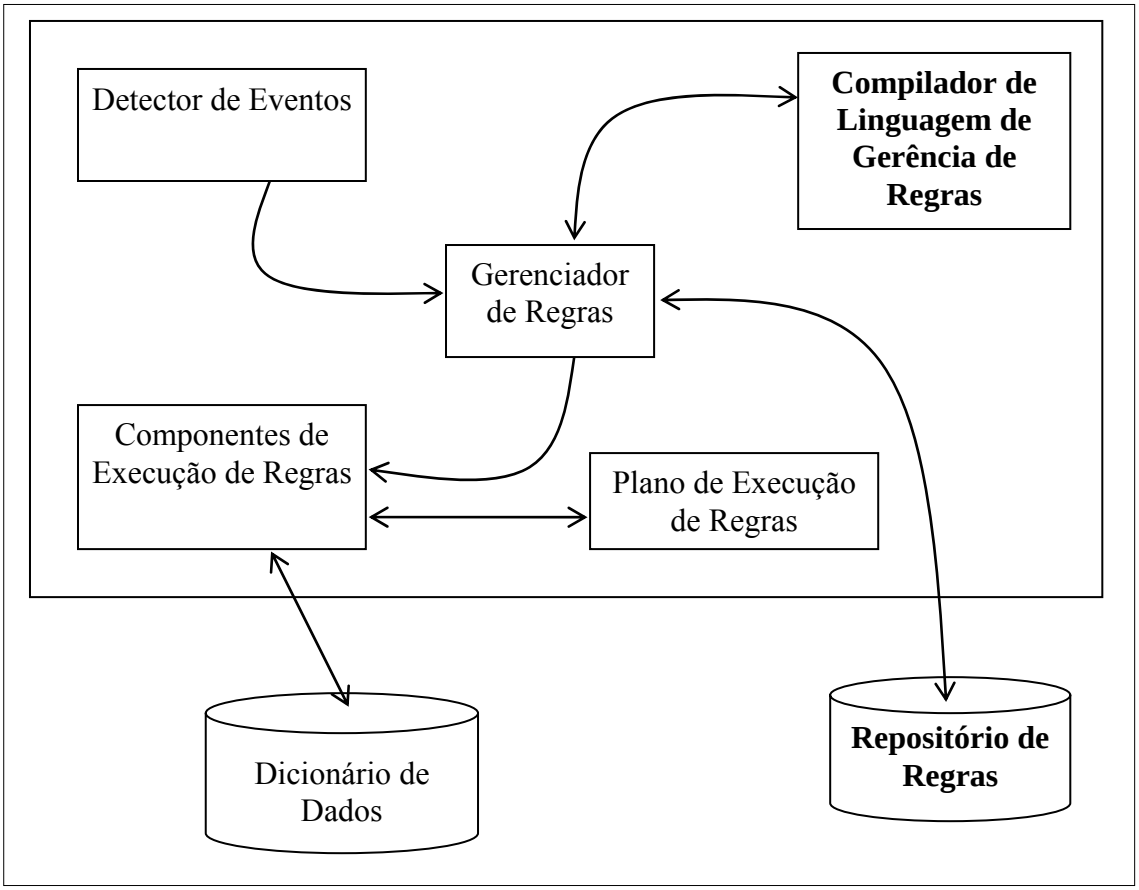

Figura 26 – Arquitetura de um SBDA genérico (adaptação de PAVON(2005))

Os componentes da arquitetura de um subsistema de execução de regras foram estendidos para se adequar à linguagem de regras adotada neste trabalho. Portanto, a descrição a seguir considera que cada componente, da arquitetura apresentada, incorpora as características do modelo adotado.

O componente "Detector de Eventos" tem a finalidade de identificar e notificar as ocorrências de eventos ao Gerenciador de Regras. Os eventos detectados são todos os eventos de banco de dados e evento temporal, sugeridos no modelo adotado. O Gerenciador de Regras tem a finalidade de armazenar e recuperar as regras do Repositório de Regras. Ele recupera as regras do Repositório sempre que existe uma notificação do Detector de Eventos, e armazena

as regras, uma vez que as regras sejam compiladas pelo “Compilador de Linguagem de Gerência de Regras”. Este Compilador tem a finalidade de compilar os tipos de regras que foram sugeridos na linguagem adotada. Considerando que a sintaxe adotada é similar à sintaxe da linguagem SQL3, então, este componente precisa ser adaptado para incorporar estas inovações. Além disso, precisa compilar as sintaxes das operações de gerência de regras sugeridas neste trabalho, uma vez que, estas operações mantêm a mesma concepção adotada na sintaxe da linguagem SQL3..

O Componente de Execução de Regras tem a finalidade de implementar o modelo de execução de regras adotado pela linguagem usada como referência neste trabalho. Este componente interage com o componente responsável pelo Plano de Execução de Regras, quando um conjunto de regras é disparado pelo mesmo evento.

O Dicionário de Dados de um SBDA possui estruturas para armazenar regras do tipo ECA e EA e seus respectivos relacionamentos, porém necessita ser adaptado para os novos tipos de regras sugeridos pela linguagem adotada neste trabalho. Este trabalho, por sua vez, sugere uma adaptação do dicionário de dados redefinindo-o em dois conjuntos de estruturas, a saber: um conjunto para armazenar dados e outro conjunto para armazenar regras. O primeiro conjunto mantém o mesmo conceito de dicionário de dados, e o segundo conjunto, é definido como repositório de regras, com estruturas propícias para o armazenamento dos tipos de regras que são sugeridas na linguagem de gerência de regras. Além disso, estas estruturas contemplam informações sobre o relacionamento entre regras. Estes dois conjuntos coexistem dentro do banco de dados de um SBDA. Portanto, o banco de dados necessita ser adaptado para tratar as estruturas oferecidas neste trabalho, sugeridas como extensão do dicionário de dados. Na Figura 26, são apresentados dois repositórios separados logicamente, porém fisicamente fazem parte da mesma estrutura, que refere-se ao banco de dados do SGBDA.

Na Figura 26, os retângulos em negrito são extensões que fazem parte da proposta desta tese.

5.5 CONCLUSÕES

Uma característica importante que deve estar presente em um repositório de regras de um SGBDA é sua capacidade de adaptar-se a mudanças ao longo do tempo. Para que isto ocorra é necessário o suporte de um modelo de regras que ofereça uma linguagem de regras semanticamente mais rica que o modelo de regras da linguagem SQL3. Isto porque a linguagem SQL3 é limitada, pois a partir dela é possível derivar somente dois tipo de regras (ECA e EA) e possui somente dois tipos de operações de gerência (criar e eliminar uma regra. Porém, o modelo adotado carece de uma infra-estrutura dotada de operações de gerência de regras e um conjunto de regras de consistência que garantam a utilização destas operações.

Este trabalho propõe esta infra-estrutura, sugerindo um conjunto de operações de gerência e um conjunto de regras de consistência, e desta forma estende~~ndo~~ a linguagem adotada aqui. Com estas operações de gerência, aliadas a este novo modelo de definição de regras, é possível dar suporte à evolução do repositório de regras.

No Manifesto de Banco de Dados Ativo proposto por um grupo de pesquisadores da área de Banco de Dados Ativos, (ACT-NET CONSORTIUM, 1996), é sugerido que o repositório de regras suporte modificações de regras, da mesma forma que os dados assumem modificações ao longo do tempo, assim como um SGBDA deve adotar um ambiente que contenha ferramentas para facilitar a inspeção do conjunto de regras armazenadas no repositório de regras, por parte do usuário, por exemplo, por meio de um navegar de regras. Este capítulo contribui para que estas sugestões descritas no manifesto se materializem, oferecendo um conjunto de operações de gerência de regras e um navegador para facilitar a gerência destas regras, por parte do usuário.

No capítulo 6 é apresentado um estudo de caso para validar a proposta deste trabalho.

CAPITULO 6. ANÁLISE DA PROPOSTA

6.1 INTRODUÇÃO

Neste capítulo, apresenta-se um estudo de caso de uma aplicação de negócio. Esta aplicação envolve dois processos de negócio que abrangem a solicitação de auxilio financeiro para participação em uma feira, denominado auxilio-evento, e o cancelamento deste auxilio-evento. Os dois processos de negócio são especificados na linguagem de gerência de regras proposta. Ambos processos de negócio envolvem regras do tipo 1 e do tipo 2.

Uma vez que os processos foram especificados, uma análise é feita sobre as regras resultantes, com a finalidade de atender a novos requisitos. Para tal finalidade, são utilizadas operações de gerência de regras.

No final do capítulo é realizada uma análise comparativa entre a linguagem de gerência de regras e a linguagem SQL3 considerando as diferentes operações de gerência de regras apresentadas neste estudo de caso.

6.2 ESTUDO DE CASO

O estudo de caso apresentado está dividido em três partes. A primeira parte descreve o contexto do negócio, a segunda parte especifica as regras de negócio na linguagem de regras adotada neste trabalho e a terceira parte a avaliação e melhorias adotadas sobre o conjunto de regras especificado, por meio de operações de gerência.

6.2.1 O Contexto do Negócio

Considere uma empresa que possui vários departamentos. Um destes departamentos tem a finalidade de realizar pesquisas sobre novos produtos e materiais. À medida que os resultados vão sendo obtidos, estes podem ser apresentados em eventos, ou os funcionários podem participar de eventos para conhecer o estado da arte sobre os temas que estiverem investigando.

A empresa oferece aos funcionários a possibilidade de participar de eventos. Para tanto, basta que o funcionário solicite recurso financeiro para seus gastos pessoais e com o translado até o evento. Este recurso financeiro necessário para a participação em eventos é

chamado de auxílio-evento. Uma vez que este auxílio foi solicitado, são verificados os pré-requisitos para esta solicitação, os quais são: o tempo que este funcionário pertence à empresa, grau de compatibilidade da área de atuação do funcionário com o foco do evento e verificar se a data de solicitação para a participação no evento ocorre antes dos trinta dias da data de início do evento. Além disso, o funcionário pode pedir um auxílio-evento para um evento, por ano, desde que o departamento ao qual ele pertence não tenha acumulado qualquer tipo de penalidade, no mesmo ano de solicitação da solicitação do auxílio. Estas penalidades ocorrem em função da desistência de funcionários, que obtiveram o auxílio, não solicitaram cancelamento dele e, também não compareceram ao evento.

Uma vez que o funcionário preencha os requisitos necessários para a solicitação do auxílio-evento, é necessário calcular o valor deste auxílio a ser atribuído a ele. O valor do auxílio é calculado em função do valor da inscrição no evento, translado até o evento e o valor do recurso outorgado pela empresa (quinze por cento do seu salário) para custeio de hotel e alimentação. Estes valores fazem parte de uma cota destinada ao departamento, e ao ultrapassar o valor desta cota, previamente estabelecida, pela empresa, para o departamento, o pedido é negado.

O funcionário, após ter solicitado o auxílio evento, deve esperar para a receber a notificação de aceitação ou rejeição deste pedido, visto que, existem cotações de preços a serem realizadas, em função dos dados da viagem.

O resultado desta solicitação de auxílio financeiro é uma mensagem do tipo “O Pedido de auxílio evento de código ”P” na data dd/mm/aaaa foi APROVADO” ou a negação da frase. Os funcionários que acumulam participações em eventos internacionais e eventos nacionais, recebem gratificações, como incentivo.

No diagrama de atividades da Figura 26 são apresentados todos os passos do processo de negócio, relativo a solicitação de auxílio-evento.

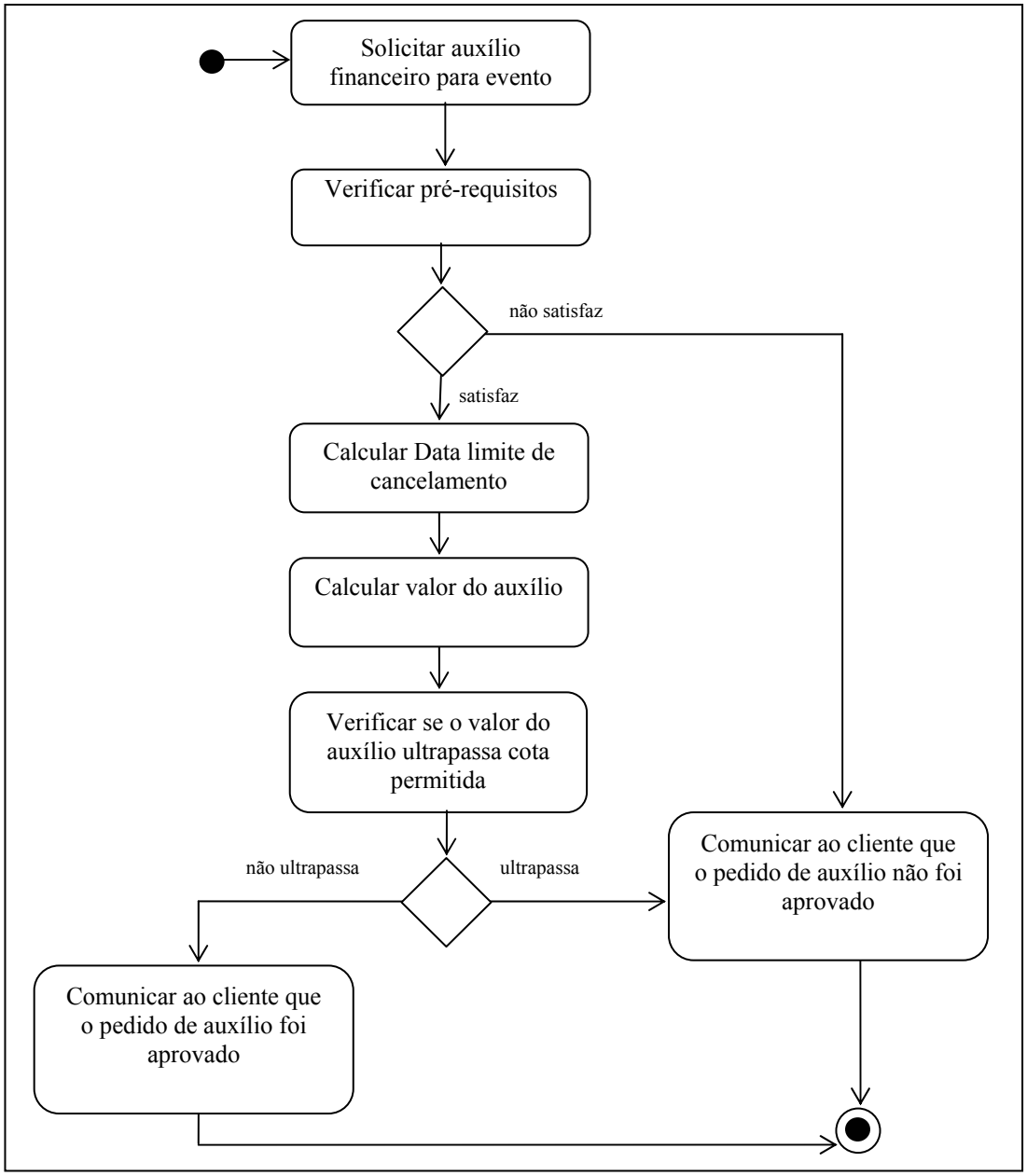

Figura 26 - Diagrama de atividades do processo de negócio “Solicitar auxílio-evento”

Considerando que um funcionário pode se deparar com eventuais problemas, sejam eles de natureza pessoal ou profissional, que podem levar um funcionário à não viajar a um determinado evento, então existe um processo de cancelamento de solicitação de auxílio-evento, chamado de cancelamento-auxílio-evento, que pode ser solicitado pelo funcionário. Este pedido pode ser solicitado somente depois que o processo de solicitação de auxílio-evento para este funcionário foi aprovado. Além disso, existem outros critérios que devem ser verificados para que o cancelamento seja efetivado. Estes critérios podem ser observados na Figura 27, e serão detalhados a seguir.

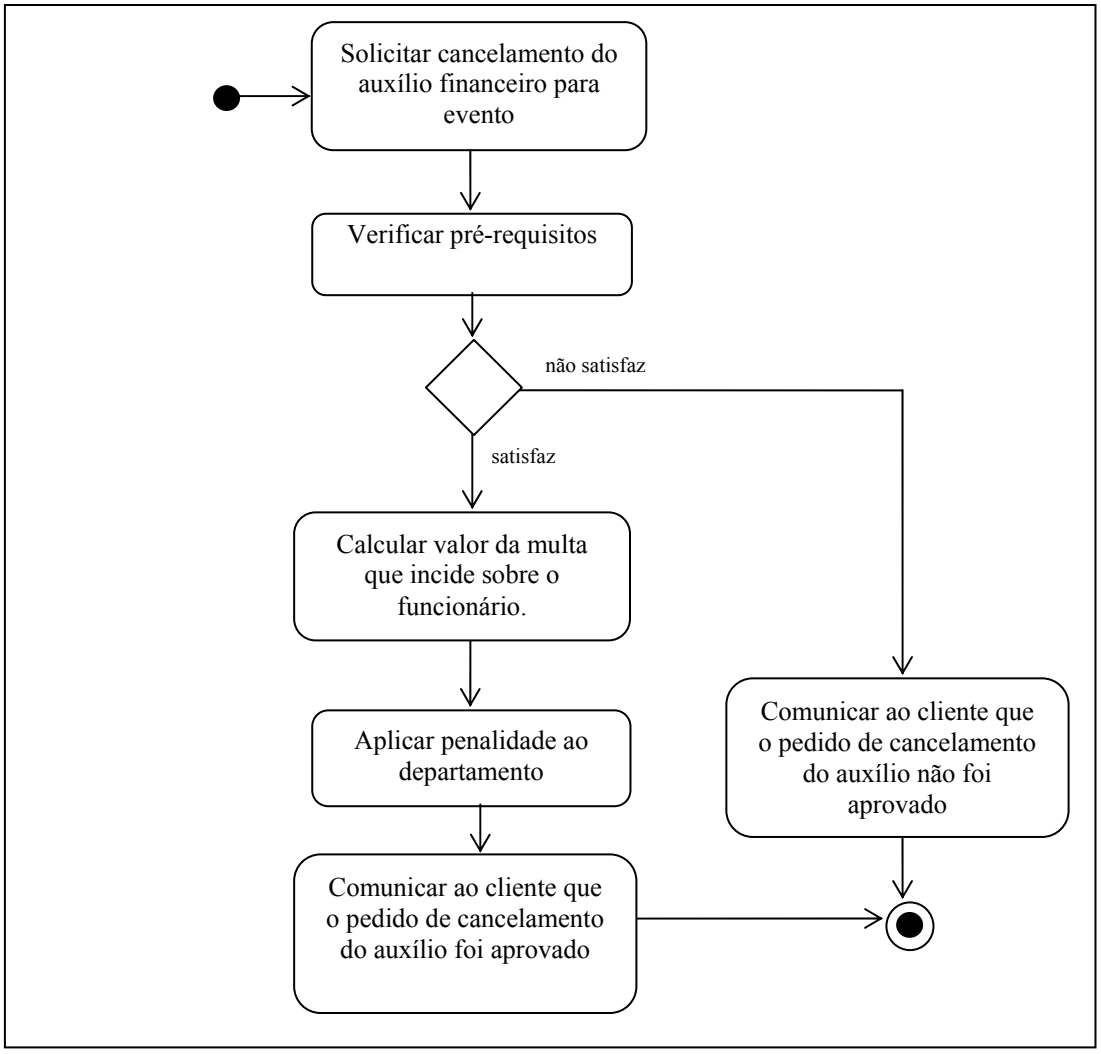

Figura 27 - Diagrama de atividades do processo de negócio “Cancelar auxílio evento”

Os critérios para o pedido de cancelamento-auxílio-evento são: verificar se o pedido deste cancelamento está dentro dos prazos estabelecidos pela empresa e calcular as multas associadas a este cancelamento, que devem ser aplicadas ao funcionário. O período de tempo para o cancelamento-auxílio-evento está entre o final da solicitação do pedido de auxílio-evento (com aprovação) e uma semana antes da data de participação neste evento. Na Figura 27 são apresentados todos os passos do processo de negócio relativo ao cancelamento de auxílio evento

Estes dois pré-requisitos, prazo e aceitação da solicitação de auxílio-evento, são necessários para que se possa calcular a multa a ser atribuída ao funcionário. A multa quando houver, deve-se a dois fatores: multa da companhia aérea pela devolução de passagens e multa por cancelamento de inscrição no evento, além da devolução do recurso financeiro para

gastos com alimentação é hotel, à empresa. Além da multa, é aplicada uma penalidade ao funcionário, e ao departamento, que não poderão participar de outro evento internacional no ano corrente..

O resultado da solicitação de cancelamento-auxilio-evento é uma mensagem do tipo "O Pedido de cancelamento do auxílio evento de código "C" na data dd/mm/aaaa foi APROVADO e o valor da multa é 'M' reais" ou a negação do pedido de cancelamento-auxilio-evento

6.2.2 As Regras de Negócio

As regras de negócio associadas aos processos de negócio de solicitação de auxílio para participação em eventos e de cancelamento desta solicitação de auxílio são descritas a seguir, de acordo com a seguinte lógica: em primeiro lugar é apresentado o esquema de dados sobre o qual os processos de negócio foram especificados. Logo a seguir, em segundo lugar, é apresentada uma breve descrição em linguagem natural da regra de negócio, para depois, ser apresentada a sua correspondente representação, na linguagem proposta.

As tabelas utilizadas para o caso prático são as seguintes:

EVENTO (codEvento, nome, entidadePromotora, dataInicio, dataFim, cidade, pais, focoEvento, valorInscricao, tipoEvento)

FUNCIONARIO (codFunc, nome, cpf, end, cep, cidade, salario, dataAdmissao, codCategoria, areaAtuacao, *coddepto*)

PEDIDO_AUXILIO (numPed, *codFunc*, *codEvento*, tipoPedido, dataSolicitaAuxilio, status, participa, valorAuxilio, dataLimCancelamento, dataSolicitaCancelamento, solicitaCancelamento)

PARTICIPA_EVENTO (*codFunc*, *codEvento*, dataParticipacao)

DEPARTAMENTO (codDepto, nome, cotaAuxilioEvento, quantPenalidades)

Obs: As chaves-primárias estão sublinhadas e as chaves-estrangeiras estão em itálico.

Os atributos abaixo possuem um domínio constituído dos seguintes valores:

- status = {Cancelado, Aprovado, NaoAprovado, Pendente}
- tipoEvento = {Nacional, internacional}
- participa = {0=não participou, 1=participou}
- penalidade = {N=não, S=sim}
- solicitaCancelamento = {N=não, S=sim}
- tipoPedido = {S= solicitação, C = cancelamento}

Descrição da regra R5:

QUANDO o funcionário solicita auxilio financeiro para um evento

ENTÃO deve-se verificar se os pré-requisitos são satisfeitos.

```
CREATE RULE R5 /*avalia pedido de auxilio financeiro
                        (estimulo-resposta / ea) */
AFTER INSERT ON PEDIDO-AUXILIO
FOR EACH ROW
DO
  BEGIN
                :V_codFunc = :new.codFunc;
                :V_codEvento = :new.codEvento;
                :V_numPed = :new.numPed;

                FIRE R10; /* verifica requisitos para pedido de auxilio-evento */

  END;
```

Na Figura 28 é ilustrado o encadeamento de regras entre R5 e R10. A regra R5 dispara a regra R10 por meio de uma operação *Fire*.

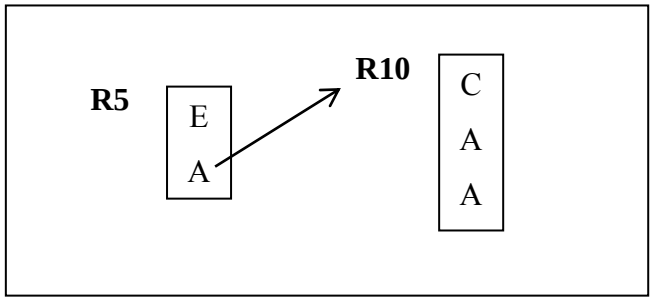

Figura 28 – Encadeamento entre a regra R5 e R10

Descrição da regra R10

SE o funcionário

- possui 24 meses de casa;
- o foco do evento é compatível com a área de atuação do funcionário;
- fez a solicitação do auxilio-evento 1 mês antes do inicio do evento;
- não haja solicitado um auxilio-evento nesse ano;
- pertence a um departamento que não apresente penalidades.

ENTÃO

- satisfaz os requisitos necessários para a solicitação de auxilio e
- portanto deve-se :
    - calcular a data limite de solicitação de cancelamento evento
    - calcular o valor do auxilio a ser atribuído ao funcionário
    - validar a viabilidade de conceder o valor do auxílio.

SENÃO

- ele não satisfaz os requisitos e deve ser comunicado desta decisão.

```
CREATE RULE R10  /* verificar os pré-requisitos para
                          auxilio-evento (Derivação - CAA)*/
WHEN ( (SELECT MONTHS_BETWEEN (SYSDATE, data_admissao)
     FROM FUNCIONARIO
     WHERE codFunc = :V_codFunc) >= 24)
AND
    (EXISTS (SELECT *
```

```
                FROM      EVENTO E, FUNCIONARIO F
                WHERE    F.codFunc = :V_codFunc
              AND     E.codEvento = :V_codEvento
              AND     E.focoEvento = F.areaAtuacao) )
AND
   ( (SELECT MONTHS_BETWEEN (SYSDATE, data_inicio)
     FROM      EVENTO
        WHERE     codEvento = :V_codEvento) >= 1)

AND
    (NOT EXISTS (SELECT *
                FROM PEDIDO_AUXILIO
                WHERE codFunc = :V_codFunc
             AND status = 'Aprovado') )
AND
    (EXITS (SELECT *
               FROM   DEPARTAMENTO D, FUNCIONARIO F
               WHERE  F.codFunc = V_codFunc
            AND    D.codDepto = F.codDepto
            AND    D.penalidade = 'N') )
DO

 BEGIN
                    UPDATE PEDIDO_AUXILIO
                    SET status = 'PENDENTE', tipoPedido = 'S'
                    WHERE numPed = :V_numPed;

     FIRE R15  /*calcular a data limite de solicitação de cancelamento do
                      auxilio-evento */
     FIRE R20; /* calcular o valor do auxilio a ser atribuído ao funcionário */
     FIRE R25   /*validar a viabilidade de conceder o valor do auxílio*/

 END;
ELSEDO
 BEGIN
                    UPDATE PEDIDO_AUXILIO
```

```
        SET status = 'NÃO APROVADO', tipoPedido = 'S'
        WHERE numPed = :V_numPed;

        callProcedure ComunicaCliente ('Auxilio Nao Aprovado');

    END;
```

Na Figura 29 é ilustrado o encadeamento de regras entre R10 e as regras R15, R20 e R25. A ação primária de R10 dispara explicitamente três regras, por meio da operação *Fire*.

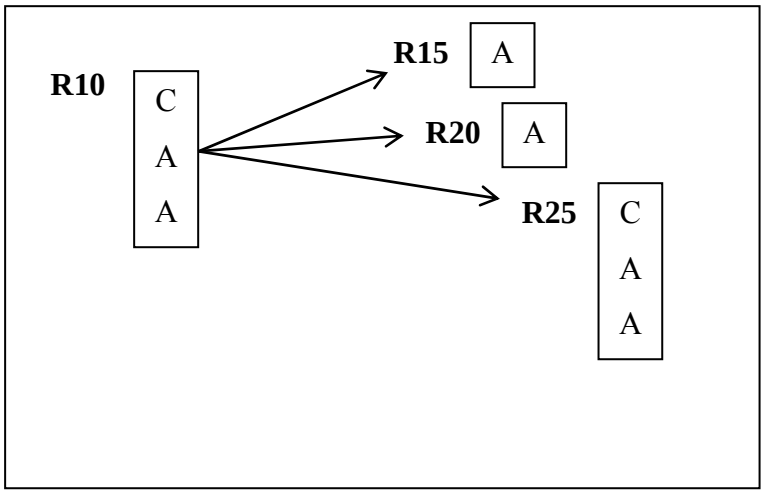

Figura 29 – Encadeamento entre a regra R10 e as regras R15, R20 e R25

Descrição da regra R15

CALCULAR a data limite para solicitar o cancelamento do pedido de auxilio ao evento. A data deve ser uma semana antes da data de inicio do evento.

```
CREATE RULE R15 /* (operacional - A) */
DECLARE
V_dataLimite EVENTO.dataInicio%type;

DO
  BEGIN
```

```
        SELECT (7 - dataInicio) INTO V_dataLimite
        FROM        EVENTO
                WHERE       codEvento = :V_codEvento;

        UPDATE PEDIDO_AUXILIO
        SET    dataLimCancelamento = V_dataLimite
        WHERE  numPed = :V_numPed;
    END;
```

Descrição da regra R20

CALCULAR o valor auxílio evento atribuído a um funcionário. O cálculo é realizado da seguinte maneira:

- Soma-se o valor da inscrição no evento com o valor do auxilio financeiro que corresponde a 15 por cento de seu salário, para custeio de hotel e alimentação.

```
    CREATE RULE R20 /* (operacional - A) */
    DECLARE
    V_valorInscricao EVENTO.valorInscricao%type;
    V_viatico FUNCIONARIO.salario%type;
    V_valorAuxilioEvento PEDIDO_AUXILIO.valorAuxilio%type;
    DO
      BEGIN

                SELECT valorInscricao INTO V_valorInscricao
                FROM EVENTO
                WHERE       codEvento = :V_codEvento;

                SELECT (salario * 0,15) INTO V_auxilio
                FROM        FUNCIONARIO
                WHERE       codFunc = :V_codFunc;

                V_valorAuxilioEvento = V_valorInscricao + V_auxilio;

                UPDATE PEDIDO_AUXILIO
                SET valorAuxilio = V_valorAuxilioEvento
        WHERE numPed = :V_numPed;
    END;
```

Descrição da regra R25

SE auxilio financeiro para deslocar-se ao evento não ultrapassar a cota financeira destinada ao departamento.

ENTÃO atualizar status e comunicar cliente que pedido aprovado.

SENÃO atualizar status e comunicar cliente que pedido não aprovado.

```
CREATE RULE 25 /* avaliar a viabilidade de conceder o valor do auxílio
                                    (Derivação - CAA) */
DECLARE
V_codDepto DEPARTAMENTO.codDepto%type;
WHEN ( (SELECT SUM(valorAuxilio)
      FROM   PEDIDO_AUXILIO
      WHERE  dataSolicitaAuxilio >= (SELECT TRUNC (SYSDATE, 'YEAR')
                                                  FROM DUAL) ) >=
        ( SELECT cotaAuxilioEvento
                  FROM DEPARTAMENTO
                  WHERE codDepto = :V_codDepto) );
DO
 BEGIN
        UPDATE PEDIDO_AUXILIO
                SET status = 'APROVADO', tipoPedido = 'S'
                WHERE numPed = :V_numPed;

                callProcedure ComunicaCliente ('Auxilio Aprovado');
          END;
ELSEDO
 BEGIN
                UPDATE PEDIDO_AUXILIO
                SET status = 'NÃO APROVADO', tipoPedido = 'S'
                WHERE numPed = :V_numPed;
                callProcedure ComunicaCliente ('Auxilio Nao Aprovado');
          END;
```

Obs: SELECT TRUNC (SYSDATE, 'YEAR') FROM DUAL esta sentença considera a data a partir de primeiro de janeiro do ano corrente.

Na Figura 30 é ilustrado o encadeamento de regras pertinentes ao processo de solicitação auxilio-evento

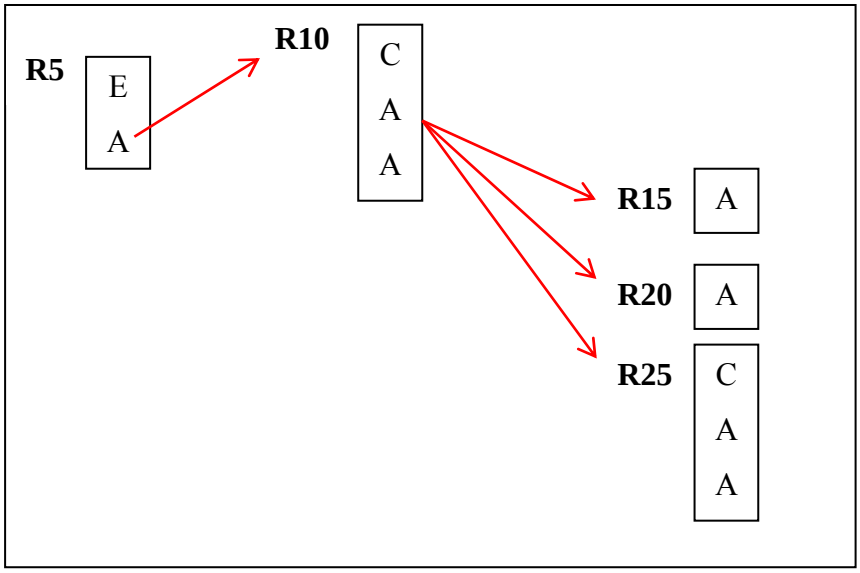

Figura 30 – Encadeamento de regras referente ao auxilio-evento

A seguir são apresentadas as regras referentes ao segundo processo de negócio: cancelamento-auxilio-evento.

Descrição da regra R30

QUANDO um funcionário solicita cancelamento de um auxilio-evento

SE solicitação do auxilio-evento foi aprovada,

ENTÃO: verificar se o funcionário satisfaz, obrigatoriamente, os pré-requisitos para este cancelamento,

SENÃO: notificá-lo de que o pedido de cancelamento foi negado.

```
CREATE RULE R30 /* (regra tipo ECAA)*/
AFTER UPDATE OF solicitaCancelamento ON PEDIDO_AUXILIO
FOR EACH ROW
WHEN (     old.status = 'Aprovado'
   AND      new.solicitaCancelamento = “S’    )
DO
```

```
BEGIN
        :V_codFunc = :new.codFunc;
        :V_codEvento = :new.codEvento;
        :V_numPed = :new.numPed;
        FIRE R35;  /* verifica requisitos para cancelamento-auxilio-evento */
END;
ELSEDO
BEGIN
        CallProcedure ComunicaCliente ('Não pode ser solicitado');
END;
```

Na Figura 31 é ilustrado o encadeamento de regras entre R30 e a regra R35. A ação primária de R30 dispara, por meio de uma operação *Fire* a regra R35.

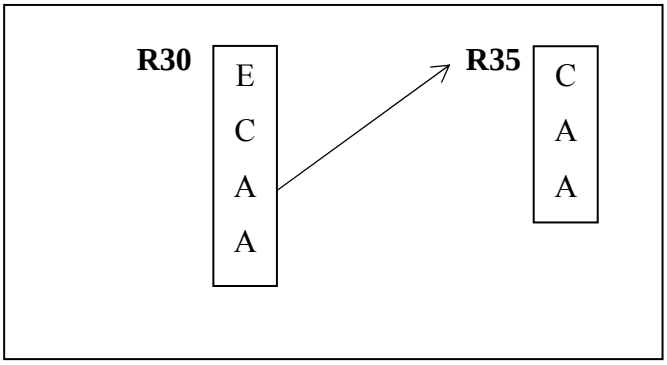

Figura 31 – Encadeamento entre a regra R30 e a regra R35

Descrição da regra R35

Esta regra verifica os pré-requisitos de cancelamento-auxilio-evento,

SE as condições dadas a seguir ocorrerem:

- data de cancelamento esta dentro do período que compreende uma semana antes do inicio do evento e depois do encerramento de solicitações d auxílio-evento.
- status para cancelamento com valor "N" (indica que não houve outra solicitação neste mesmo período, para o mesmo evento).

ENTÃO calcula-se a MULTA a ser atribuída ao funcionário, como também

- atribui-se uma penalidade ao departamento

SENÃO: notificar que a solicitação não foi aprovada.

```
CREATE RULE R35   /* verificar os pre-requisitos para
                                    cancelamento-evento (Derivação - CAA)*/
WHEN (( SELECT  PA.codEvento
                FROM    PEDIDO_AUXILIO PA, EVENTO E
                WHERE   PA.solicitaCancelamento = 'N'
                  AND   PA.dataSolicitaCancelamento > PA.dataSolicitaAuxilio
                  AND   PA. dataSolicitaCancelamento < (7 - E.dataInicio)
                  AND   PA.numPed = :V_numPed);
DO
  BEGIN
                UPDATE PEDIDO_AUXILIO
                SET status = 'Cancelado', tipoPedido = 'C'
                WHERE codEvento = :current.codEvento
                AND codFunc = :current.codFunc;

        FIRE R40   /* calcular multa a ser atribuída ao funcionário */
                FIRE R45   /* atribuir penalidade ao departamento */
  ELSEDO
    BEGIN
                CallProcedure ComunicaCliente ('Cancelamento não aprovado');
    END;
```

Na Figura 32 é ilustrado o encadeamento de regras entre a regra R35 e as regras R40 e R45. A regra R35 possui duas ações, primária e secundária. Na ação primária, são disparadas por meio da operação *Fire*, duas regras, e uma regra por meio de uma operação de banco de dados.

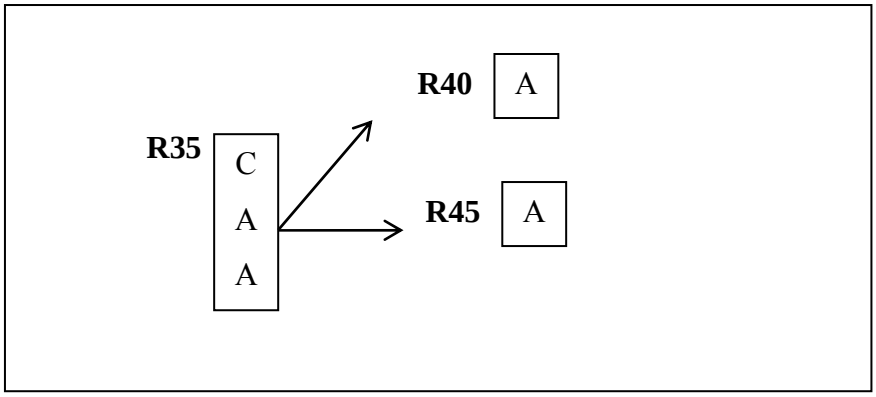

Figura 32 - Encadeamento entre a regra R35 e as regras R40 e R45

Descrição da regra R40

O valor da Multa a ser atribuída (repassado) a um funcionário é calculado da seguinte maneira:

CALCULAR:

- calculo da multa por devolução de passagem aplicada pela companhia aérea;
- calculo da multa por cancelamento de inscrição no evento; e
- devolução dos 15% do salário atribuídos a auxilio financeiros.

CREATE RULE R40 /*(cálculo do valor da multa – (Operacional - A)*/
DO
BEGIN
CallProcedure calculaMultaFuncionario( );
END;

Obs: O procedimento calculaMultaFuncionário representa o cálculo da multa a ser atribuída ao funcionário. Esta multa é a somatória dos valores cobrados pela devolução da passagem ao evento, do cancelamento ao evento e devolução dos recursos financeiros utilizados para gastos com hospedagem e alimentação.

Descrição da regra R45

ATRIBUIR penalidade ao departamento em função da desistência de algum funcionário deste departamento (Esta penalidade implica que o departamento não poderá participar de nenhum evento internacional no ano seguinte à aplicação da pena).

```
CREATE RULE R45 /*(Atribuir penalidade – (Operacional - A)*/
DO
  BEGIN

        CallProcedure penalidadeDepartamento( );

        UPDATE DEPARTAMENTO
        SET    penalidade = 'S'
        WHERE  codFunc = :V.codFunc;

  END;
```

Na Figura 33 é apresentado o encadeamento de regras relacionado com o processo de negócio para cancelar um auxilio evento.

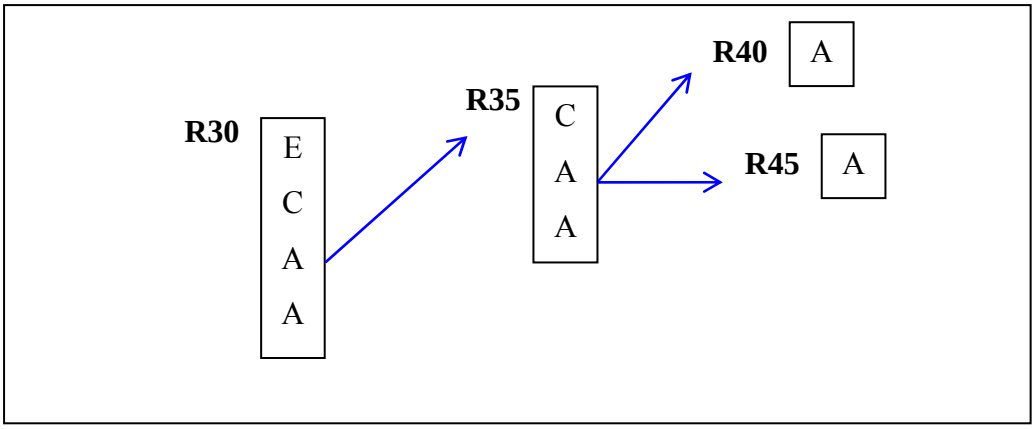

Figura 33 - Encadeamento de regras referente ao cancelamento-auxilio-evento

Descrição da regra R50

QUANDO um funcionário acumula um numero de participações em eventos,

ENTÃO recebe gratificações em seu salário.

```
DECLARE
V_quantidade            numeric;
CREATE RULE R50  /* Gratificações regra tipo EA */
AFTER INSERT ON PARTICIPA-EVENTO
FOR EACH ROW
DO
  BEGIN

                SELECT COUNT(dataParticipacao) INTO :V_quantidade
                FROM   PARTICIPA_EVENTO
                WHERE  codFunc = :new.codFunc;

                IF V_quantidade > 10 THEN
                        UPDATE FUNCIONARIO
                        SET    salario = salario + (salario * 0,05)
                        WHERE  codFunc = :new.codFunc;
        ENDIF;
END;
```

Os processos de negócio solicitação e cancelamento de auxilio-evento resultam em dois conjuntos de regras, encadeadas, e armazenadas no repositório de regras. Neste repositório de regras, é possível observar que existem regras encadeadas por meio de operações *Fire* e por meio de operações de banco de dados. Da mesma forma, existe uma regra que não faz parte de nenhum encadeamento, regra R50. O encadeamento de regras, referente ao primeiro processo de negócio é representado pelas flechas vermelhas e o segundo encadeamento de regra é formado pelas flechas azuis. As regras R5, R30 e R50 são regras do tipo estímulo-resposta e seus eventos, assim como o encadeamento de regras referentes aos dois processos de negócio são ilustrados na Figura 34

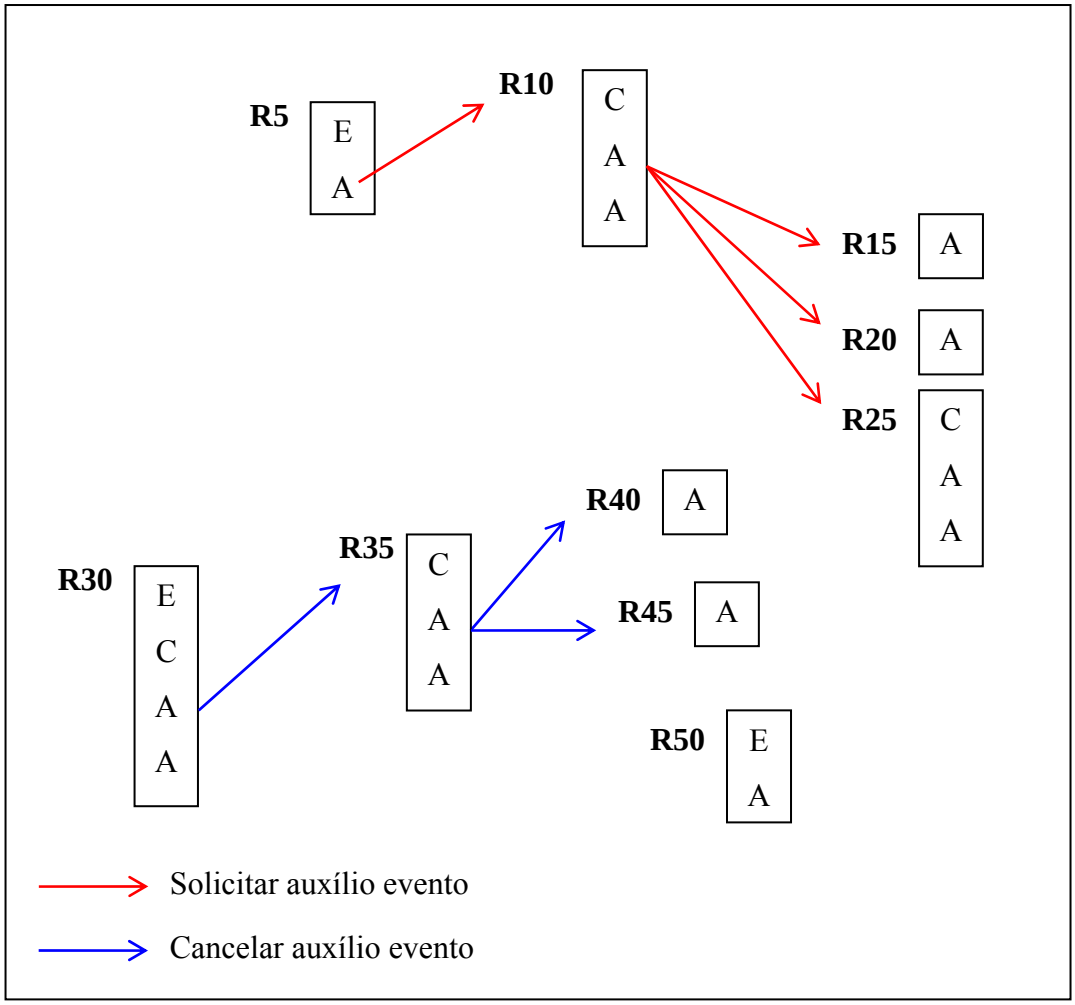

Figura 34 - Encadeamento de regras do estudo de caso

6.2.3 As Operações de Gerência

O estudo de caso apresentado na seção anterior é composto por dois conjuntos de regras. Um conjunto para solicitar auxilio-evento e outro conjunto para cancelamento desta solicitação. Considerando que as regras refletem o comportamento dinâmico do negócio, então mudanças nas especificações deste negócio, conseqüentemente, levam a mudanças nas regras. O impacto destas mudanças, no repositório de regras, é analisado neste estudo de caso. Primeiro em função das operações de gerência e, segundo, em função das operações de gerência sugeridas pela linguagem SQL3. Estas mudanças se apresentam como conseqüência de novas especificações no processo de negócio e também em função das melhorias que se pode dar ao conjunto de regras armazenadas no repositório de regras, sem mudar a semântica apresentada pelos processos de negócio, que elas representam. Este processo é chamado de

refatoração de regras.

- Uma situação possível de acontecer, referente às modificações no negócio, ocorre quando a empresa não tem mais interesse em automatizar o processo de cancelamento de auxilio-evento. Este controle será feito manualmente, com a funcionária da empresa, visto que a empresa passa a valorizar a entrevista, do funcionário que solicitou o auxilio-evento, com a funcionária responsável pelo processo de cancelamento deste auxilio. Para evitar que sejam eliminadas as regras que representam o cancelamento, optou-se por desabilitá-las. A vantagem de desabilitar estas regras é que no futuro, a empresa pode solicitar a reativação do processo de cancelamento, e desta forma, estas regras já estão disponíveis para a função desejada.

Para evitar que se desabilite uma regra por vez, optou-se por agrupá-las e depois desabilitá-las. Portanto, primeiro as regras são agrupadas e depois desabilitadas.

Seja a seguinte operação de gerência:

DEFINE RULESET cancelamento-auxilio-evento
ADD RULE R30, R35, R40, R45;

Esta operação agrupa as regras R30, R35, R40 e R45. O seguinte passo é desabilitar este grupo de regras.

Seja a seguinte operação:

DISABLE RULESET cancelamento-auxilio-evento**;**

Esta operação desabilita o conjunto de regras definido como “cancelamento-auxilio-evento”

- **Análise da linguagem SQL3**

O repositório de regras contém regras que representam dois processos de negócio, auxílio-evento e cancelamento-auxilio-evento. O segundo processo sofreu modificações, devido às mudanças nas especificações. Foi solicitado que as regras pertinentes ao segundo processo de negócio fossem eliminadas. A solução inicial foi desativá-las, visto que futuramente poderiam ser reaproveitadas, de forma independente, ou o processo de

cancelamento-auxilio-evento ser utilizado novamente. Esta solução foi possível com o auxilio das operações de gerência.

Considerando uma solução com as operações disponíveis na linguagem SQL3, somente é possível eliminar as regras relacionadas ao cancelamento-auxilio-evento. Da mesma forma, estas regras não podem ser reaproveitadas. Para voltar a usar o processo de cancelamento-auxilio-evento, é necessário criar novamente essas regras. Como a linguagem SQL3 não possui outros tipos de regras, além de ECA e EA, todas as regras a serem definidas são do tipo estímulo-resposta, como conseqüência, estas regras ao serem criadas, novamente, são associadas a um valor de "tempo de criação", utilizado como critério para executar as regras que são disparadas pelo mesmo evento. Portanto a ordem de criação das regras deve ser considerada pelo usuário. Quanto maior o número de regras a serem criadas, maior a complexidade inerente a este processo de criação.

- Uma outra situação que deve ser tratada neste repositório é a refatoração das regras, que foram definidas no repositório de regras, e que possuem parte do código, repetido. As regras R10 e R25 possuem parte do código repetido e que por sua vez, podem ser refatoradas em uma única regra, que contenha estes códigos, sem perder a semântica relativa ao processo de negócio auxilio-evento. A regra R10 possui em sua ação secundária uma chamada a uma função, cuja finalidade é alertar o usuário que o processo de solicitação-evento foi cancelado. Este mesmo procedimento existe na regra R25, em sua ação secundária. Na ação primária desta regra, existe uma chamada a procedimento que alerta o usuário de que o processo de solicitação auxilio-evento, foi aceito.

Considerando as duas regras, R10 e R25, é possível converter esta lógica em uma única regra, definida como regra R27, conforme é apresentado a seguir:

Descrição da regra R27

Esta regra tem a finalidade de notificar ao funcionário que o seu processo de solicitação de auxilio-evento, ou cancelamento-evento foi aprovado ou cancelado, pela empresa. Quando o processo é do tipo auxílio-evento, então a regra R27 dispara a regra R28, que avalia se o processo iniciado pelo funcionário foi aprovado ou reprovado. Quando o

processo é do tipo cancelamento-evento, a regra R27 dispara a regra R29, que avalia o pedido de cancelamento foi aprovado.

QUANDO o funcionário, solicita auxilio-evento.
SE: tipo de pedido está relacionado com a solicitação auxilio-evento
ENTÃO: verificar se solicitação foi aprovada ou reprovada
SENÃO: verificar se cancelamento foi aprovado.

```
CREATE RULE R27 /*notificação do pedido de auxilio-evento e
                                    cancelamento (estimulo-resposta / ECAA) */
AFTER INSERT OR UPDATE OF tipoPedido  ON PEDIDO-AUXILIO
FOR EACH ROW
WHEN ( :new.tipoPedido = 'S')      /*S = solicitação*/
DO
  BEGIN
            FIRE   R28 */notifica o status de auxilio pedido –
                        Aprovado ou Reprovado(CAA)*/
  END;
ELSEDO
  BEGIN
            FIRE   R29 */notifica o status de cancelamento pedido –
                        Aprovado(CA)*/
   END;
```

Esta regra R27 é do tipo ECAA e compõe-se de duas regras R28 e R29. O evento acontece quando o usuário solicita auxilio-evento, ou cancelamento do auxilio-evento.

A condição da regra verifica se o tipo de pedido é “Auxilio-evento”. Caso afirmativos, a regra R28 é disparada, caso contrário, a regra R29 é disparada.

Na Figura 35 é apresentado o encadeamento de regras, incluindo a regra R27, R28 e R29.

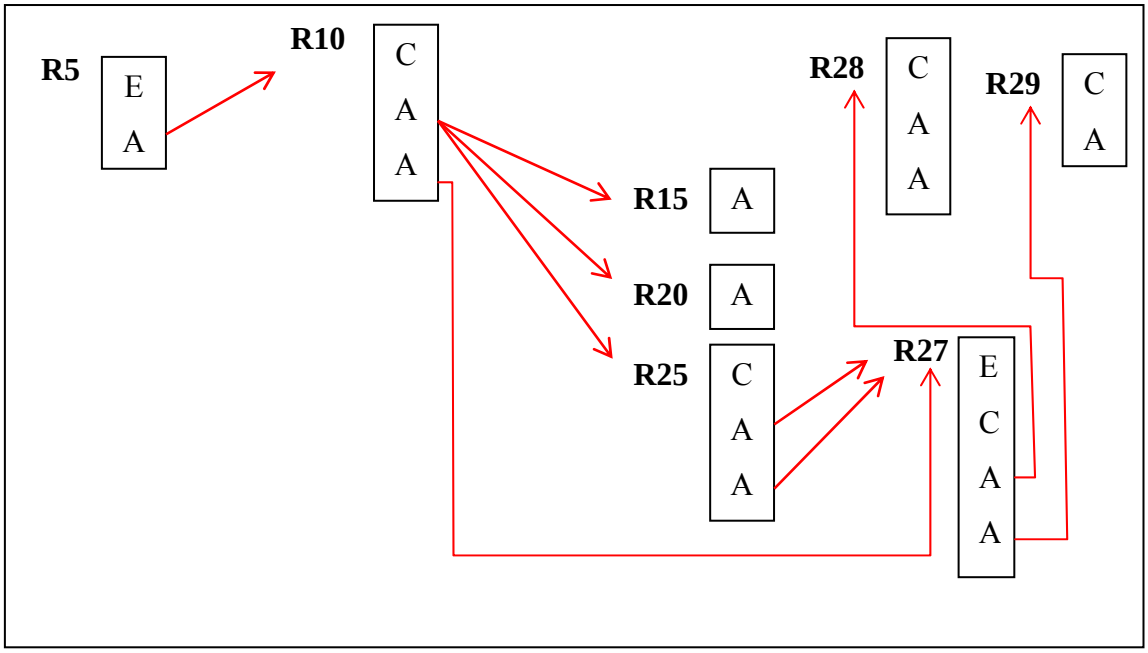

Figura 35 - Encadeamento de regras considerando a inclusão da regra R27.

As definições das regras R28 e R29 são:

Descrição da regra R28

Esta regra tem a finalidade de notificar ao funcionário que o seu processo de solicitação de auxilio evento foi aprovado.

Cada vez que um funcionário solicita auxilio financeiro para um evento, este funcionário deve ser alertado automaticamente, pelo sistema, de que o processo de avaliação foi concluído.

QUANDO o funcionário, solicita auxilio-evento.
SE: status do pedido de solicitação for aprovado
ENTÃO: avisar que solicitação foi aprovada
SENÃO: avisar que solicitação não foi aprovada.

```
CREATE RULE R28 */notifica o status de auxilio pedido –
                                Aprovado ou Reprovado(CAA)*/
WHEN ( :current.status = 'APROVADO')
DO
  BEGIN
```

```
                callProcedure ComunicaCliente ('Auxilio Aprovado');

  END;

ELSEDO
  BEGIN
                callProcedure ComunicaCliente ('Auxilio Nao Aprovado');
  END;
```

Descrição da regra R29

Esta regra tem a finalidade de notificar ao funcionário que o seu processo de cancelamento-auxilio-evento foi aprovado.

Cada vez que um funcionário solicita o cancelamento do auxilio-evento, este funcionário deve ser alertado automaticamente, pelo sistema, de que o processo foi concluído.

QUANDO o funcionário, solicita cancelamento-auxilio-evento.
SE: status do pedido de cancelamento-auxilio-evento for aprovado
ENTÃO: avisar que cancelamento foi aprovado

```
CREATE RULE R29 */notifica o status de cancelamento auxilio pedido –
                                Aprovado ou Reprovado(CA)*/
WHEN ( :current.status = 'CANCELADO')
DO
  BEGIN
                callProcedure ComunicaCliente ('Solicitação cancelada');
  END;
```

Para inserir a regra R27 no contexto do auxilio-evento, é necessário definir as regras R27, R28 e R29, alterar a ação secundária da regra R10 e alterar a ação primária e secundária da regra R25. Estas operações são apresentadas a seguir:

A operação para a definição da regra R27, R28 e R29 foram apresentadas anteriormente. Restam apenas as alterações na regra R10 e R25. Para alterar a regra R10, utiliza-se a seguinte operação:

```
ALTER RULE <R10>
MODIFY SECONDARY ACTION ELSEDO
<  BEGIN
                UPDATE PEDIDO_AUXILIO
                SET status = 'NÃO APROVADO'
                WHERE numPed = :V_numPed , tipoPedido = 'S';

          END;
>;
```

As alterações na regra R25 são as seguintes:

```
ALTER RULE R25
MODIFY PRIMARY ACTION TO
  BEGIN
      UPDATE PEDIDO_AUXILIO
            SET status = 'APROVADO', tipoPedido = 'S'
            WHERE numPed = :V_numPed;
  END;

ALTER RULE R25
MODIFY SECONDARY ACTION TO
  BEGIN
            UPDATE PEDIDO_AUXILIO
            SET status = 'NÃO APROVADO', tipoPedido = 'S'
            WHERE numPed = :V_numPed;
   END;
```

Estas operações modificam, primeiro a ação secundária da regra R10, depois a ação primária da regra R25 e logo a seguir a ação secundária da regra R25. Uma vez que estas ações, primária e secundária, foram modificadas, é necessário compilar e lincar com suas respectivas regras, que estão armazenadas no repositório de regras.

O impacto desta mudança, no repositório de regras pode ser analisado em função do encadeamento ilustrado na Figura 36. A regra R35, do tipo derivação, possui uma ação primária e uma ação secundária. Na ação primária é definida uma operação de banco de dados, que leva ao disparo da regra R27. Considerando que a regra R27 é utilizada para notificações, então a mesma foi estendida para o processo de cancelamento-auxilio-evento.

Esta organização auxilia a manutenção destas regras.

- **Análise da linguagem SQL3**

A refatoração destas regras com a finalidade de melhorar a legibilidade e compreensão do código, torna-se mais difícil de ser aplicada quando a abordagem envolve somente as operações sugeridas pela linguagem SQL3. Isto porque, esta limitação está associada ao conjunto de operações oferecidas, criação e eliminação de regras. Portanto, para o caso apresentado, seria necessário eliminar e voltar a definir as regras R10, R25 e R35. Da mesma forma, deveriam ser definidas as regras R27, R28 e R29. Considerando que os tipos de regras em questão são ECA e EA, e que ao defini-las lhes é atribuído um tempo de criação, e que este tempo de criação é utilizado para o critério de seleção de regras do conjunto de regras disparadas simultaneamente, então a quantidade de regras em questão, dificulta a manutenção do repositório de regras, no que tange a refatoração de regras.

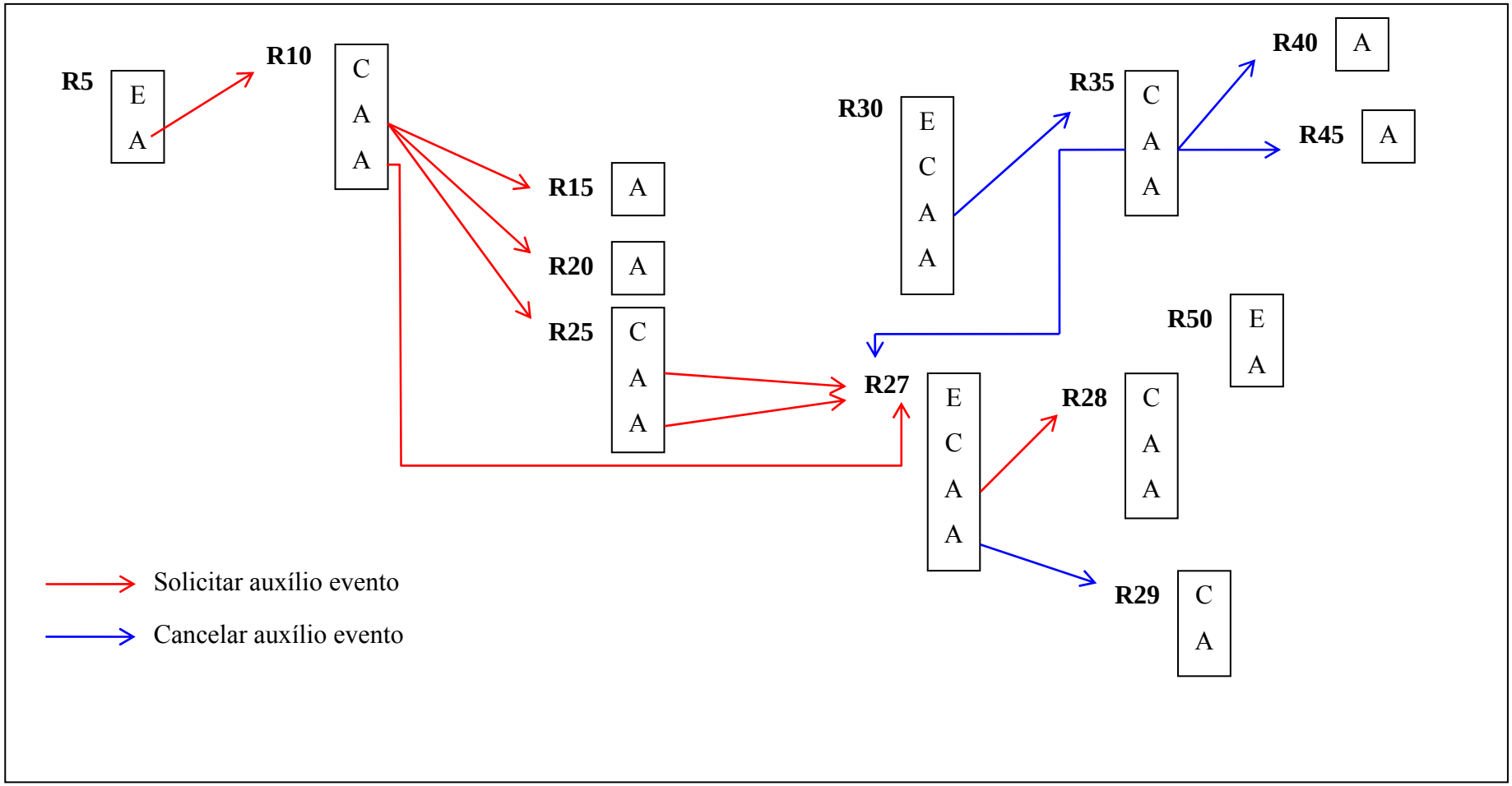

Figura 36 - Encadeamento de regras, considerando a inclusão da regra R27 no repositório de regras.

Para minimizar o número de regras ilustrado no encadeamento de regras da Figura 36, e também, para melhorar a legibilidade destas regras, é realizada a fusão das regras R30 e R35. A seqüência de operações que devem ser seguidas, para a regra R35, são: adicionar um evento, modificar a condição e modificar a ação primária. Quanto à regra R30, esta deve ser excluída.

Para adicionar um evento utiliza-se a seguinte operação:

ALTER RULE R35
ADD EVENT UPDATE OF solicitaCancelamento ON PEDIDO_AUXILIO
ACTIVATION TIME AFTER
GRANULARITY FOR EACH ROW;

Quando a granularidade e o tempo de ativação não forem especificados, estes valores são atribuídos automaticamente, pelo SGBDA. Para o tempo de ativação, o valor padrão é *AFTER* e para a granularidade, o valor padrão é *FOR EACH ROW*.

O resultado desta operação é a seguinte sentença:

AFTER UPDATE OF solicitaCancelamento ON PEDIDO_AUXILIO
FOR EACH ROW

Para modificar a condição da regra, utiliza-se a seguinte operação:

ALTER RULE R35
MODIFY CONDITION TO

```
        (SELECT  PA.codEvento
                FROM    PEDIDO_AUXILIO PA, EVENTO E
                WHERE   PA.solicitaCancelamento = 'N'
                  AND   PA.dataSolicitaCancelamento > PA.dataSolicitaAuxilio
                  AND   PA. dataSolicitaCancelamento < (7 - E.dataInicio)
                  AND   PA.numPed = :V_numPed)
AND
                (     old.status = 'Aprovado' AND new.solicitaCancelamento = “S’)
                );
```

Para modificar a ação primária da regra, utiliza-se a seguinte sentença:

```
ALTER RULE R35
MODIFY PRIMARY ACTION TO
BEGIN
        :V_codFunc = :new.codFunc;
        :V_codEvento = :new.codEvento;
        :V_numPed = :new.numPed;

        UPDATE PEDIDO_AUXILIO
        SET status = 'Cancelado', tipoPedido = 'C'
        WHERE codEvento = :current.codEvento
        AND codFunc = :current.codFunc;

    FIRE R40   /* calcular multa a ser atribuída ao funcionário */
        FIRE R45   /* atribuir penalidade ao departamento */
END;
```

A regra R35 passa a ter a seguinte definição:

```
CREATE RULE R35   /* verificar os pre-requisitos para
                          cancelamento-evento (Derivação - CAA)*/
AFTER UPDATE OF solicitaCancelamento ON PEDIDO_AUXILIO
FOR EACH ROW

WHEN (( SELECT  PA.codEvento
        FROM    PEDIDO_AUXILIO PA, EVENTO E
        WHERE   PA.solicitaCancelamento = 'N'
          AND   PA.dataSolicitaCancelamento > PA.dataSolicitaAuxilio
          AND   PA. dataSolicitaCancelamento < (7 - E.dataInicio)
          AND   PA.numPed = :V_numPed)
AND
        (     old.status = 'Aprovado' AND new.solicitaCancelamento = “S’)
        )

DO
  BEGIN
        :V_codFunc = :new.codFunc;
```

```
                    :V_codEvento = :new.codEvento;
                    :V_numPed = :new.numPed;

                    UPDATE PEDIDO_AUXILIO
                    SET status = 'Cancelado', tipoPedido = 'C'
                    WHERE codEvento = :current.codEvento
                    AND codFunc = :current.codFunc;

            FIRE R40   /* calcular multa a ser atribuída ao funcionário */
                    FIRE R45   /* atribuir penalidade ao departamento */
        END
        ELSEDO
          BEGIN
                    CallProcedure ComunicaCliente ('Cancelamento não Aprovado');
          END;
```

A operação para eliminação da regra R30 é apresentada abaixo.

DROP RULE R30;

- **Análise da linguagem SQL3**

As operações aplicadas à regra R35 foram pontuais, ou seja, alterou-se apenas o necessário. As operações de gerência disponíveis na linguagem SQL3, permitem apenas a definição e exclusão das regras. O fato de definir a regra R35, implica que é necessário avaliar se não existem outras regras com o mesmo evento, pelo fato de que ela pode ser disparada em conjunto com outras regras.

O repositório de regras resultante, após as modificações realizadas, com a finalidade de melhorar a legibilidade das regras, é ilustrado na Figura 37.

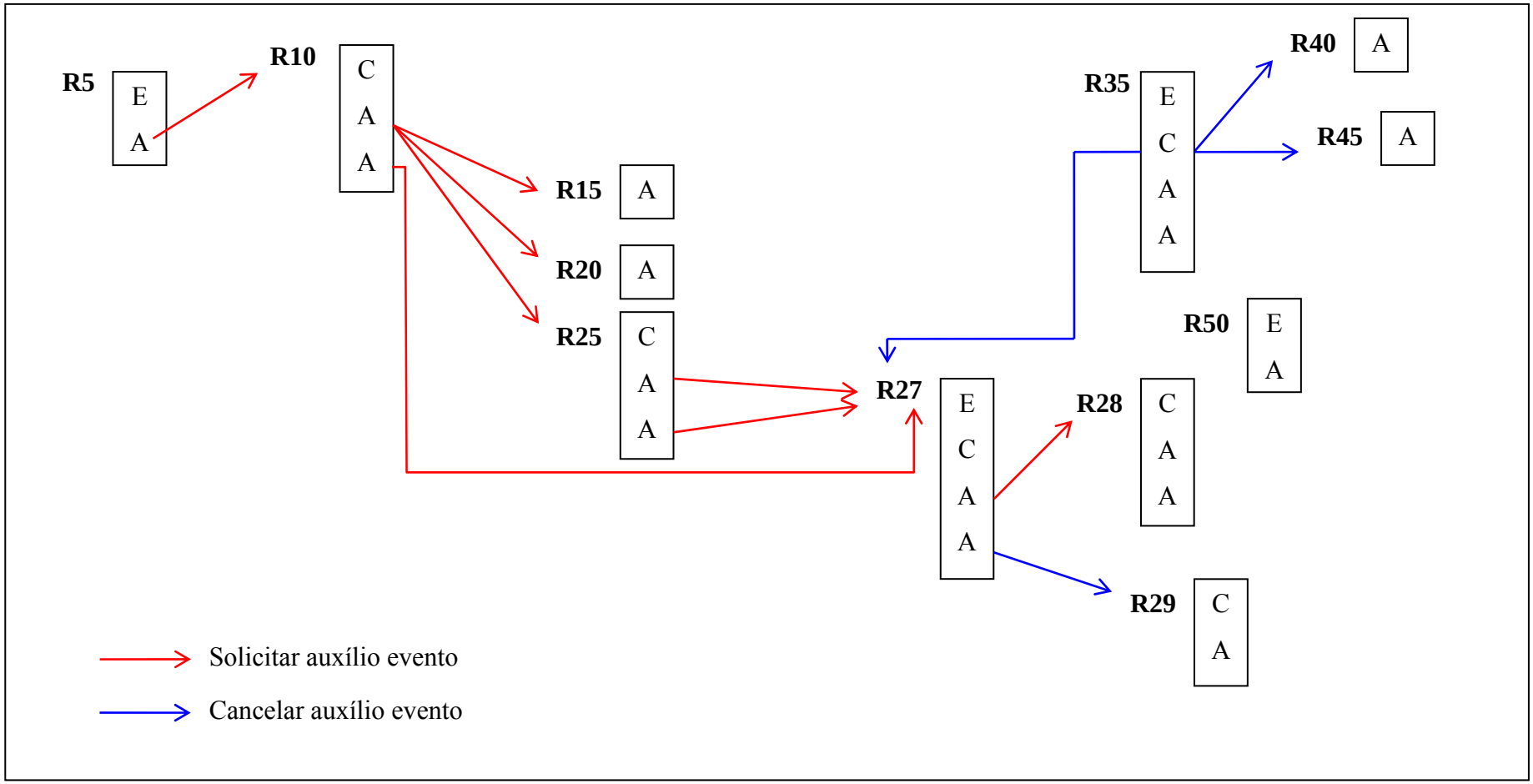

Figura 37 - Encadeamento de regras, após as alteração na regra R35 e eliminação da regra R30.

6.3 CONCLUSÕES

A finalidade deste capítulo foi apresentar um estudo de caso por meio do qual foi possível aplicar as operações de gerência sugeridas neste trabalho. Considerando que as regras definidas no repositório de regras necessitavam adaptar-se a um requisito de negócio, foram utilizadas operações que suprissem estas mudanças, sem a necessidade de eliminá-las do repositório. Da mesma forma, houve a necessidade de realizar manutenção nas regras, refatorando-as. Nesta refatoração foi reutilizado código de outras regras, e estenderam-se algumas regras que faziam parte do repositório.

Este processo de modificação do repositório de regras para adaptar-se a dinâmica de processos de negócio, por meio da adição de regras, exclusão de regras, alteração de regras, e de suas partes constitui-se na evolução do repositório de regras. Esta evolução, de forma ágil, foi possível de ser realizada com a linguagem de gerência de regras proposta.

No capítulo seguinte são apresentadas a conclusão do trabalho e futuras pesquisas que podem ser realizadas em função da linguagem apresentada.

CAPITULO 7. CONCLUSÕES E PESQUISAS FUTURAS

7.1 PRINCIPAIS CONTRIBUIÇÕES

Os SBDAs possuem um dicionário de dados para armazenar informações sobre o esquema de banco de dados e regras. Além destas estruturas, eles possuem uma linguagem de gerência de regras composta por operações para criação de regras e eliminação das mesmas. Estas operações, e as estruturas utilizadas nestes SGBDAs seguem o modelo sugerido pela SQL3. O conjunto de operações não é flexível o suficiente para facilitar a evolução do dicionário de dados, da mesma forma que o próprio dicionário de dados não oferece flexibilidade para o armazenamento de regras e seus relacionamentos. Esta flexibilidade é fundamental para prover o usuário de mais um ambiente em que ele pode armazenar e gerenciar suas regras computacionais.

As atuais aplicações, que manipulam grandes quantidades de dados e regras, necessitam de um ambiente que suporte o gerenciamento de dados e regras. Os dados já dispõem de uma infra-estrutura adequada para o seu armazenamento e gerenciamento, o que não ocorre com as regras. Com relação às regras, é necessário um ambiente que garanta, de forma consistente, a evolução do repositório que as mantém.

Esta evolução somente é possível com um conjunto de operações de gerência de regras tendo como base um repositório de regras e uma linguagem de gerência de regras estendida, visto que o atual dicionário de dados da linguagem SQL3 não oferece os recursos necessários para armazenar, eficientemente, informações sobre regras. Portanto, uma das contribuições desta proposta, dentro da área de regras em banco de dados, é a **identificação das limitações da linguagem SQL3 em relação às operações de gerência de regras**. Tendo conhecimento destas limitações, é possível compreender quais são as operações que podem ser aproveitadas e quais devem ser estendidas para permitir uma representação mais apropriada das operações de gerência.

Este trabalho, ao propor uma adequação do paradigma de linguagem de gerência de regras, da linguagem SQL3, estendeu-o e adaptou-o com o propósito de adequá-lo às necessidades atuais das aplicações. Esta proposta de adequação representa uma mudança no conceito de armazenamento de regras, sugerido pela linguagem SQL3. Os dados e regras coexistirão em um mesmo ambiente, tal como era antes, porém cada um deles dispõe de um conjunto de estruturas adequadas à sua realidade. Neste sentido, o dicionário de dados auxilia

o armazenamento dos dados, tendo um conjunto de estruturas para garantir a consistência dos dados, por meio das restrições. As regras também passam a contar com uma infra-estrutura, denominada repositório de regras, cuja finalidade é armazenar as regras, e dispõe de um conjunto de regras de consistência para manter a corretude do repositório de regras.

Os dicionários de dados que compunham tanto dados quanto regras foram analisados com a finalidade de identificar seus aspectos positivos e suas respectivas deficiências. Até esta etapa do trabalho, dados e regras coexistem em um mesmo ambiente, sob o domínio de uma linguagem de definição de dados. Esta linguagem é utilizada para definir tabelas, regras entre outros objetos do banco de dados.

Na busca por oferecer uma linguagem de gerência de regras com maior expressividade que a linguagem SQL3, e que pudesse oferecer recursos para as necessidades das atuais aplicações, foi analisado o modelo de definição de regras da linguagem proposta por Pavón (2005). Esta linguagem teve como premissa estender o modelo de regras da linguagem SQL3, definindo um conjunto de tipos de regras, com maior expressividade, que os tipos sugeridos pela linguagem SQL3.

Uma outra contribuição deste trabalho é a **realização de uma avaliação crítica do modelo de definição de regras,** com a finalidade de identificar questões que não foram tratadas pelo modelo adotado, tais como: operações para a alteração de regras, operações para alterar partes de uma regra e análise de um repositório de regras para armazenar os diferentes tipos de regras propostos.

A partir desta análise, identificaram-se as características dos tipos de regras oriundos do modelo adotado. Estas características serviram de base para a elaboração do repositório de regras, além disso, as informações sobre as operações de gerenciamento de regras completaram a análise necessária para identificar as estruturas iniciais, que servem para armazenar as regras.

Além destas estruturas, foi identificado um conjunto de meta-regras que deveriam existir para garantir a consistência das informações que eram manipuladas no repositório de regras. Estas situações inconsistentes poderiam ocorrer em função do uso das operações de gerência, como por exemplo, adicionar uma ação secundária a uma regra que não possui condição (EA). As estruturas são tabelas que armazenam informações sobre as regras, suas partes e o relacionamento entre elas. Este relacionamento representa o mapeamento correspondente ao encadeamento de regras, que surge em função de um conjunto de regras

caracterizadas pelas regras do tipo estímulo-resposta. Além disso, são armazenadas informações sobre a composição de regras e a conseqüente prioridade definida entre elas.

Portanto, uma das contribuições deste trabalho consiste na **extensão do repositório de regras da linguagem SQL3**. Estender o repositório de regras representa especificar um conjunto de estruturas adicionais para armazenar os tipos de regras adotados neste trabalho. Estas estruturas adicionais, focadas nas regras, contemplam um outro conjunto de regras, necessárias para manter a sua consistência.

A partir da concepção do repositório de regras, foram elaboradas operações de gerência de regras. Estas operações são utilizadas para criar uma regra, eliminar uma regra, alterar parte de uma regra. A **extensão das operações de gerência de regras** passa a ser outra contribuição valiosa. Estender as operações de gerência de regras implica em prover a linguagem de regras adotada neste trabalho de um conjunto significativo de operações para criação, alterações e exclusões de regras, com a finalidade de facilitar a manutenção de regras, e de suas partes. Esta extensão aliada à extensão do repositório de regras, habilita este repositório a **garantir a evolução do mesmo**. Esta evolução é acompanhada de mecanismos que garantem a sua consistência.

Considerando que nem todo usuário domina ou conhece a linguagem de gerência de regras, foi sugerido um navegador de regras, cuja finalidade é permitir a edição de um conjunto de regras existente no repositório de regras. Desta forma, o navegador provê assistência ao usuário, de modo que este possa se beneficiar das funcionalidades de uma ferramenta que serve para navegar sobre o repositório de regras. Esta **proposta de um protótipo para navegação sobre o repositório de regras** é mais uma contribuição da tese, pois a evolução do repositório de regras deve vir acompanhada de ferramentas que dêem suporte ao usuário, facilitando-lhe o uso do repositório de regras por meio de uma interface apoiada com usabilidade.

7.2 FUTURAS PESQUISAS

A especificação de uma linguagem de gerência de regras em SBDA é uma contribuição importante para que o usuário tenha outra opção, flexível, em se tratando de armazenamento e gerência de regras de negócio. Para que esta proposta se materialize, é necessário que seja completado um conjunto de tarefas. Estas tarefas incluem aspectos

relacionados à execução, e armazenamento de regras, bem como ferramentas de apoio ao usuário.

Este trabalho abordou os aspectos relativos à especificação de um repositório de regras e uma linguagem de gerência de regras, porém é necessário que outros aspectos sejam considerados, gerando assim outras linhas de pesquisas. Dentre as pesquisas futuras sugere-se a **adaptação do repositório de regras a um SGBDA comercial ou acadêmico**. Esta adaptação implica na elaboração de um conjunto de estruturas para armazenar regras. Estas estruturas podem ser estendidas a qualquer outro ambiente que suporte o gerenciamento de regras, como por exemplo, ambiente para *workflows*. Além desta adaptação, é importante que o SGBDA interprete a operações de gerência, realizadas pelo usuário ou por uma ferramenta, portanto um outro trabalho que pode ser realizado é o **desenvolvimento de um compilador da linguagem de gerência de regras**, considerando as meta-regras sugeridas. Uma vez que o compilador esteja implementado, análises de desempenho da linguagem poderiam ser verificadas.

A elaboração de uma ferramenta para auxiliar ao usuário no gerenciamento de regras, armazenadas no repositório de regras, tal como um **navegador de regras**, alinhado aos conceitos de usabilidade, é uma linha de pesquisa que pode ser desenvolvida como mais um trabalho futuro, principalmente porque a expressividade da linguagem de regras melhorou, quando comparada com a linguagem SQL3, e os trabalhos sugeridos nesta linha, considera somente o modelo de regras da linguagem SQL3.

Tendo em vista que o repositório de regras conterá inúmeras regras, é importante verificar se existem ciclos indesejados no repositório. Estes ciclos ocorrem em função de disparos implícitos e disparos explícitos. Uma outra linha de pesquisa é **desenvolver um analisador de regras**, identificar as propriedades de terminação e confluência de regras em conjunto de regras, ou seja, verificar a existência destes ciclos infinitos de regras e também verificar se determinados grupos de regras são confluentes. Ciclos infinitos são tratados na literatura como “terminação de regras”. Um conjunto de regras é dito confluente se a partir de um estado inicial, existem vários caminhos que levam a um mesmo estado final de execução.

Considerando que existem **regras no formato XML**, então o repositório pode ser estendido e propor outras operações de gerência para este tipo de regras, assim é possível armazenar e gerenciar estes tipos de regras de negócio, usado em intercambio eletrônico de informações, por meio de documentos XML.

REFERÊNCIAS BIBLIOGRÁFICAS

ABITEBUL, S.; BENJELLOUN, O.; MILO,T. Positive active XML. In: ACM SIGMOD-SIGACT-SIGART SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS (PODS), 23, Paris, 2004. **Proceedings**. ACM Press, New York, 2004, p.14-16.

ACT-NET CONSORTIUM. The active database management system manifesto: a rulebase of ADBMS features. **ACM SIGMOD Record**, v.25, n.3, p.40-49, Sept. 1996.

AIKEN, A.; WIDOM, J.; HELLERSTEIN, J. M. Static analysis techniques for predicting the behavior of database production rules. **ACM Transactions on Database Systems (TODS)**, v.20, n.1, p.3-41 1995

AIKEN, A.; HELLERSTEIN, J. M ;WIDOM, J. Behavior of database production rules: termination, confluence and observable determinism. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, San Diego, 1992. **Proceedings**. New York: ACM Press, 1992. p.59-68.

AMGHAR, Y.; MEZIANE, M.; FLORY, A. Using business rules within a design process of active databases. **Journal of Database Management**, v.11, n.3, p.3-15, 2000.

BAJEC, M.; KRISPER, M. Issues and challenges in business rule-based information systems development In: EUROPEAN CONFERENCE ON INFORMATION SYSTEMS, 30, Regensburg, Germany **Proceedings** (Bartmann D, Rajola F, Kallinikos J, Avison D, Winter R, Ein-Dor P, Becker J, Bodendorf F, Weinhardt C eds.) (cópia eletrônica), Regensburg, Germany, 12p. 2005.

BAILEY, J.; POULOVASSILIS, A.; NEWSON, P. A dynamic approach to termination analysis for active database rules. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL LOGIC, 1, London, 2000. **Proceedings**. London: Springer-Verlag, 2000. p. 1106-1120.

BARALIS, E.; CERI, S.; PARABOSCHI, S.; Compile-time and runtime analysis of active behaviors. **IEEE Transactions on Knowledge and Data Engineering**, v.10, n.3, p.353-370, May, 1998.

BARALIS, E.; WIDOM, J.; An algebraic approach to static analysis of active database rules, **ACM Transactions on Database Systems**, v.25, n.3, p.269-332, Sept, 2000.

BERNDTSSON, M. Management of rules in object-oriented databases. In: BALTIC WORKSHOP ON NATIONAL INFRASTRUCTURE DATABASES: PROBLEMS, METHODS AND EXPERIENCES. Vilnius, Lithuania. 1994. **Proceedings**. Vilvius, v.1, May, 1994, p.78-85.

BERNDTSSON, M.;MELLIN, J.; HÖGBERG, U.; Visualization of the composite event detection process. In: USER INTERFACES TO DATA INTENSIVE SYSTEMS, Edinburgh, Scotland. **Proceedings**. Washington, IEEE Computer Society Press, 1999. p. 118-128

BONATTI, P.A.; SHAHMEHRI, N.; DUMA, C.; et al. **Rule-based policy specification: state of the art and future work**, Project deliverable D1, Working Group I2, EU n.E REWERSE. 2004. (Technical Report - IST-2004-506779). Disponível em: *http://rewerse.net/deliverables/i2-d1.pdf*. Acesso em: 15 abr. 2007.

BONIFATI, A.; CERI, S.; PARABOSCHI, S. Active rules for XML: a new paradigm for E-services **The VLDB Journal** v.10, p.39–47, Aug. 2001.

BUSINESS RULES GROUP. GUIDE **Business rules project**. Final Report, Revision 1.3. Business Rules Community. 2000. Disponível em <*http://www.dbpd.com/vault/9809darc.html*> Acesso em: 12 fev. 2005.

BUTLERIS, R.; KAPOCIUS, K. The business rules repository for information systems design. In: CONFERENCE ON ADVANCES IN DATABASES AND INFORMATION SYSTEMS (ADBIS 2002), 6. Bratislava, **Proceedings,** Darmstadt, Digital Library Forum, 2002. p.64-77.

BRY, F.; ECKERT, M.; Twelve theses on reactive rules for the web. In. PROCEEDINGS OF THE WORKSHOP ON REACTIVITY ON THE WEB (EDBT Workshop 2006). Munich, Germany, **Proceedings**. Springer, Munich 2006. p-842-854

CERI, S.; COCHRANE, R.J.; WIDOM, J. Practical applications of triggers and constraints: successes and lingering issues. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE, 26., Cairo, 2000. **Proceedings**. San Francisco: Morgan Kaufmann, 2000. p.254- 262.

CERI, S.; MANTHEY, R; **Consolidated specification of chimera**. Dipartamento di Elettronica e Informazione, Politecnico di Milana, Milano, Technical Report. 84p, [IDEA.DE.2P.006.01, ESPRIT Project 6333], Nov. 1993.

CERI, S.; RAMAKRISHNAN, R.; Rules in database systems. **ACM Computing Surveys**, v.28, n.1, p.109-111, Mar. 1996.

CERI, S.; WIDOM, S.; Deriving production rules for constraint maintenance In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES, 16, Brisbane. Australia **Proceedings** Morgan Kaufmann Publishers, San Francisco. 1990. p.566-577.

CERI, S.; WIDOM, S.; Production rules in parallel and distributed database environments. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE (VLDB), 18, Vancouver, Canada **Proceedings**, Morgan Kaufman pubs. (Los Altos CA), Vancouver, 1992, p.339-351.

CHAKRAVARTHY, S. et al. Composite events for active databases: semantics, contexts and detection. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE(VLDB), 20, Santiago, Chile **Proceedings**, Morgan Kaufmann Publishers, San Francisco, 1994. p.606-617.

CHAKRAVARTHY, S.; MISHRA, D. **Snoop: an expressive event specification language for active databases**. Gainesville, Florida: Department of Computer and Information Sciences, University of Florida, 34p. [Technical Report UF-CIS-TR-93-007]. Mar. 1993

CHAKRAVARTHY, S.; TAMIZUDDIN, Z.; ZHOU, J. A visualization and explanation tool for debugging ECA rules in active databases. In: INTERNATIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS, 2, Athens, Greece. **Proceedings**, Springer-Verlag, London, 1995. p.197-212.

COLLET, C.; COUPAYE, T.; SVENSEN T.; NAOS: efficient and modular reactive capabilities in an object-oriented database system. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE(VLDB), 20, Santiago, Chile **Proceedings**, Morgan Kaufmann Publishers, San Francisco, 1994. p.132-143.

COUCHOT, A.; Improving active rules termination analysis by graphs splitting, In: EAST - EUROPEAN CONFERENCE ON ADVANCES IN DATABASE AND INFORMATION SYSTEMS (ADBIS), 5., Vilnius, Lithuania, 2001. **Proceedings.** London: Spring- Verlag. 2001. p.64-77.

DIAZ, O.; JAIME, A.; Depuración de disparadores en bases de datos activas. In: JORNADAS DE INGENIERÍA DEL SOFTWARE Y BASES DE DATOS (JISBD), 4. Cáceres **Proceedings** Escuela Politecnica, Universidad de Extremadura. 2000. p.151-162

DIAZ, O.; PATON, N.; GRAY, P.; Rule management in object oriented database: a uniform approach. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE (VLDB), 17, Barcelona. **Proceedings.** Morgan Kaufmann, 1991. p.317-326.

DITTRICH, K. R. et al. **SAMOS in hindsight: experiences in building an active object-oriented DBMS**. 31p. Department of Computer Science, University of Zurich, [Technical Report]. May. 2000.

ELIZONDO, A. J.; **Reglas activas: soporte y manejo en las bases de datos orientadas a objetos**. 1998, 193p. Tese (Doutorado) – Departamento de Lenguajes y Sistemas informáticos. Universidad del País Basco, San Sebastián. 2005.

ELMASRI, R.; NAVATHE, S.B. **Sistemas de banco de Dados**. 4.ed. São Paulo, Pearson-Addison Wesley, 2005. 724p.

ENGSTRÖM, H; BERNDTSSON, M.; LINGS, B. **ACOOD essentials**, Department of Computer Science, University of Skövde, 26p. [technical Report HS-IDA-TR-97-010]. Oct.1997.

FORS, T.; Visualization of rules behavior in active databases. In: WORKING CONFERENCE ON VISUAL DATABSE SYSTEMS (VDB-3) – IFIP2.6, 3, Skövde, Sweden **Procedings**. Chapman & Hall, London 1995. p.215-231.

FORTIER, P. SQL3:Implementing the SQL Foundation Standard. Ed. McGrow-Hill, 1999. 414p.

GATZIU, S. The SAMOS active DBMS prototype. In: HELLENIC CONFERENCE IN INFORMATICS, 6, Atenas, Grecia, **Procedings**. Atenas: Greek Computer Society, 1997.

GEPPERT, A.; GATZIU, S.; DITTRICH, K. R. **Rulebase evolution in active object-oriented database systems: adapting the past to future needs**. Zurich: Department of Computer Science, University of Zurich, 23p. 1995. [Technical Report 95.13]. 1995.

GEPPERT, A.; GATZIU, S.; DITTRICH, K.R.; FRITSCHI, H.; VADUVA, A.; **Architecture and Implementation of the active object-oriented database management system SAMOS**. Zürich, Institut für Informatik, 39p. 1995a. [Technical Report 95.29 a].1995a

GUERRINI, G. **An active and deductive object-oriented data model**. 1998. 280p. Tese (Doutorado) – Dipartimento di Informatica e Scienze dell'Informazione, University of Genova. Genova. 1998

GUISHENG, Y.;QUN, L.; JIANPEI, Z.; JIE, L.; DAXIN, L.; Petri based analysis method for active database rules. **IEEE International Conference on Systems, Man, and Cybernetics**, v.2, p.858-863, Oct. 1996

HANSON, E.N. The design and implementation of the ARIEL active database rule system. **IEEE Transaction on Knowledge and Data Engineering**, v.8, n.1, p.167-172, Fev. 1996.

HERBST, H.; MYRACH. T. A repository system for business rules. In: IFIP TC-2 WORKING CONFERENCE ON DATA SEMANTICS: DATABASE APPLICATIONS SEMANTICS, 6, London**. Procedings**, Chapman & Hall London, 1995. p.119-139.

HEUSER, C. A. **Projeto de banco de dados**.4ª ed. Porto Alegre, Sagra Luzzatto. 2004. 236.

ILOG RULES. **Business Rules**. 2005. Disponível em: <*http://www.ilog.com/*> Acesso em: 12 May. 2007.

ISO/IEC 9075-2:1999. **Information technology – database languages – SQL – Part 2: foundation (SQL/Foundation)**. 1151p. 1999. New York: American National Standards Institute - NASI, 1999.

JAIME, A.E. **Regras activas: soporte y manejo en las bases de datos orientadas a objetos.** 1998. 193p. Tese (Doutorado) – Departamento de Lenguajes y Sistemas Informáticos. Universidad del País Vasco. 1998

KANGSABANIK, P. **On transaction modeling and concurrency control in active DBMS**, 1998. 180p. Tese (doutorado) – Department of Computer Science & Engineering, Indian Institute of Technology. Kharagpur. 1998.

KAPPEL, G.; KRAMLER, G.; RETSCHITZEGGER, W.; TriGS debugger – a toll for debugging active database behavior. In: INTERNATIONAL CONFERENCE ON DATABASE AND EXPERT SYSTEMS APPLICATIONS. **Proceedings** LNCS, Springer, Munich, Germany, 2001. p-410-421.

KAPPEL, G.; RETSCHITZEGGER, W.; The TriGS active object-oriented database system – an overview; **ACM SIGMOD Record**, v.27, n.3, Sept. 1998. p.36-41.

LANG, P. A. et al. Graphical editor for the conceptual design of business rules. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 14, Orlando, **Proceedings** IEEE Computer Society, Washington. 1998. p.599.

LEMKE, T.; MANTHEY, R.; **The passive rule design tool: tool documentation**. 26p. 1997. University of Bonn, Department of Computer Science III. [Technical Report. IDEA.DE.22.O.010]. Mar. 1997.

MENS, K et al. Tools and environments for business rules In: EUROPEAN CONFERENCE ON OBJECT-ORIENTED PROGRAMMING (ECOOP - Workshop Report), 12. Belgium. **Proceedings.** Programming Technology Lab, Belgium, 1998. p.189-196.

MORGENSTERN, M.; Active database as a paradigm for enhanced computing environments. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASE (VLDB), 9, Florence, Italy **Proceedings** Morgan Kaufmann Publishers San Francisco. 1983. p. 34-42.

MORIARTY, T.; Business rule management facility, **Database Programming & Design**, Sept. 1998. Disponível em <*http://www.dbpd.com/vault/9809darc.html*> Acesso em: 12 Fev. 2005.

ORACLE CORPORATION. Oracle10g database reference, Release 1 Disponível em <*http://www.oracle.com/technology/documentation/index.html*> Acesso em: 12 May. 2007.

PATON, N. W.; DÍAZ, O.; **Active database systems**. University of Manchester ACM Computing Surveys. V.31, n.1, p.63-103. Mar. 1999.

PATON, N. W. **Active rules in database systems**, Ed. Springer Verlag, New York, USA. 1998, p.439.

PAVON, J.; **Representação de estruturas de conhecimento em sistemas de bancos de dados**, 1996. 119p. Dissertação (Mestrado). Escola Politécnica da Universidade de São Paulo, São Paulo. 1996.

PAVON, J.; VIANA, S.; CAMPOS, E.G.L.; **Análise da linguagem SQL3 com relação à especificação de regras de negócio**, In: CONFERÊNCIA LATINO AMERICANA DE INFORMÁTICA, 2006, Santiago do Chile. CLEI'06, Santiago do Chile, 2006. p.1-10. 1 CD-ROM.

PAVON, J.; **Um modelo de regras para sistemas de banco de dados ativos**. 2005. 126p. Tese (Doutorado) Departamento de Engenharia de Computação e Sistemas Digitais. Escola Politécnica da Universidade de São Paulo. São Paulo. 2005.

POSTGRESQL. **Postgresql documentation,** Version 8.0.1. 2005 Disponível em *http://www.postgresql.org/docs/8.0/static/index.html*. Acesso em: 04 jan. 2005.

RODRIGUES, J. P.; CORREIA, J. O repositório XML do SIGDiC. In: CONFERENCIA NACIONAL SOBRE XML: APLICAÇÕES E TECNOLOGIAS ASSOCIADAS, 3, Casa da Torre, Vila Verde, Braga. (XATA2005).**Acta da Conferência,** s/n, Fev, 2005.

ROSS, R. G.; **Principles of the rules business approach.** 1ed. New York. Ed. Addison Wesley, 2003. 400p.

RULEMACHINES. **Business rule studio,** 2000. Disponível em <http://www.rulemachines.com>. Acesso em: 12 Fev. 2005.

STONEBRAKER, M. The Integration of rule systems and database systems. **IEEE Transaction on Knowledge and Data Engineering**, v.4, n.5, p.415-423, Oct. 1992.

STONEBRAKER, M.; HEARST, M.; POTAMIANOS, S. A commentary on the POSTGRES rules system. **ACM SIGMOD Record** v.18, n.3, p.5-11, Sept.1989.

STONEBRAKER, M.; WOODFILL, J.; ANDERSEN, E. **Implementation of Rules in Relational Database Systems** EECS Department, University of California, Berkeley, 10p. 1983. [Technical Report UCB/ERL M83/54]. 1983.

TORRICO, F.N.; TANAKA, A. K.; MOURA, A.M.C. Especificação de regras de negócio para banco de dados relacional-objeto. In: CONFERÊNCIA LATINOAMERICANA DE INFORMATICA (CLEI), 26. Cidade de México, 2000. **Anais**. Monterrey - Mexico. (meio eletronico), 2000. 13p.

TÜRKER, C. Schema evolution in SQL-99 and commercial (Object-) relational DBMS. In: WORKSHOP ON FOUNDATIONS OF MODELS AND LANGUAGES FOR DATA AND OBJECTS, Dagstuhl Castle, Germany, Sept. 2000. **Proceedings**. Springer-Verlag Berlin, 2001. p.163-181.

TÜRKER, C.; GERTZ, M., Semantic integrity support in SQL:1999 and Commercial (Object-) Relational Database Management System. **VLDB Journal**, v.10, p.241-269, Jun. 2001.

VADUVA, A. **Rule Development for active database ystem**. 1999. 163p. Tese (Doutorado) - Department of Informatics at the University of Zurich, University of Zurich. Zurich, 1999

VADUVA, A.; GATZIU, S.; DITTRICH, K. R. Investigating termination in active database systems with expressive rule languages. In: INTERNATIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS, 3, Skovde, Sweden **Proceedings**. Springer-Verlag London, 1997. p.149-164.

VANHATALO, J.; KOEHLER, J.; LEYMANN, F. Repository for business processes and arbitrary associated metadata, In: INTERNATIONAL CONFERENCE ON BUSINESS PROCESS MANAGEMENT (BPM2006), 4, Vienna,. **Proceedings**. Lucius & Lucius, UTB, Vienna, 2006. p.25-31.

VASILECAS, O.; LEBEDYS, E. Moving business rules from system models to business rules repository, **INFOCOMP-Journal of Computer Science**.v.5, n.2, p.11-17, Jun. 2006.

VIANA, S.; PAVON, J.;ALMEIDA JR., J.R. Rule management in database systems. In: INTERNATIONAL CONFERENCE ON COMPUTING (CIC'06), 15, México, 2006. **Proceedings**. Cidade do México, IEEE Computer Society Press, 2006. p.315-322.

VON HALLE, B.; **Business rules applied**. New York. Ed. John Wiley & Sons, 2002. 546p.

WIDOM, J. The Starburst active database rule system, **IEEE Transaction on Knowledge and Data Engineering**. v.8, n.4, p.583-595, Aug. 1996.

ZIMMER, D.; UNLAND, R.; MECKENSTOCK, A. Rule termination analysis based on petri nets. In: WORKSHOP ON ON DATABASES: ACTIVE AND REAL-TIME**,** Maryland **Proceedings** ACM Press,New York, 1996. p.29-32.

ANEXO A – Resumo das Sintaxes Proposta.

1. Alterar o evento de uma regra

ALTER RULE <nome-regra>
MODIFY EVENT TO <identificação da operação> [{<OR identificação da operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;

2. Eliminar o evento de uma regra

ALTER RULE <nome-regra>
DROP EVENT <identificação da operação> [{<OR identificação da operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;

3. Adicionar um evento a uma regra

ALTER RULE <nome-regra>
ADD EVENT <<identificação da operação> [{<OR identificação da operação> <objeto que sofre a ação>}] |
<identificação da data_hora>;
[ACTIVATION TIME <identificação do tempo de ativação>]
[GRANULARITY <identificação da granularidade>];

4. Adicionar uma condição a uma regra

ALTER RULE <nome-regra>
ADD CONDITION [<operador unário>] <elemento - condição>
[<conector> <elemento - condição>] **...**

5. Eliminar a condição de uma regra

ALTER RULE <nome-regra>
DROP CONDITION;

6. Alterar a condição de uma regra

ALTER RULE <nome-regra>
MODIFY CONDITION TO [<operador unário>] <elemento - condição>
[<conector> <elemento - condição>] **...**

7. Adicionar uma ação a uma regra

ALTER RULE <nome_regra>

ADD SECONDARY ACTION <ação secundária>

8. Modificar a ação de uma regra

ALTER RULE <nome_regra>

MODIFY [PRIMARY | SECONDARY] ACTION <ação primaria | secundária >;

9. Eliminar a ação de uma regra

ALTER RULE <nome_regra>

DROP [PRIMARY | SECONDARY] ACTION;

10. Alternar as ações de uma regra

ALTER RULE < nome_regra >

CHANGE ACTION FROM [PRIMARY] TO [SECONDARY];

11. Habilitar ou Desabilitar uma regra

ENABLE [OR DISABLE] RULE < nome_regra >;
DISABLE RULE < CATEGORIA_CLIENTE>;

12. Agrupamento de regras

CREATE RULESET < nome_grupo_regra >

ADD RULE [lista de regras**]**

[OR DELETE RULE [lista de regras**]]**

13. Eliminar Grupo de Regras

DROP RULESET <nome_grupo_regra>;

14. Eliminar uma regra

DROP RULE <nome_regra>;

15. Habilitar ou Desabilitar um Conjunto de Regras

ENABLE [OR DISABLE] RULESET< nome_grupo_regra >;