Suporte ao Teste de Sistemas de Informação Baseados em Regras Ativas Escritas em SQL[1]

Virgínia Mara Cardoso[2], Plínio de Sá Leitão Júnior[3], Mario Jino[2]
[2]Universidade Estadual de Campinas – UNICAMP
Av Albert Einstein, 400, Campinas SP 13083970, Brazil
{vcardoso,jino}@dca.fee.unicamp.br
[3]Universidade Federal de Lavras - UFLA
PO Box 3037, Lavras MG 37200000, Brazil
psleitao@dcc.ufla.br

Abstract. *Active rules define actions on active databases; they are typically used for the maintenance of the database consistency as well as for the functionalities implementation in Information Systems (ISs). A tool was built to test ISs based on active rules written in SQL – ART-TOOL (Active Rule Testing TOOL). Structural software testing techniques were extended to support the use of testing criteria based on data flow analysis. The tool was used to test 15 active rules and the results are promising: all faults were revealed; and the tool supported the criteria application and the criteria coverage evaluation.*

Resumo. *Regras ativas definem ações sobre um banco de dados ativo, sendo tipicamente usadas para a manutenção da consistência da base de dados e para a implementação de funcionalidades em Sistemas de Informação (SI). Uma ferramenta foi construída para testar SIs baseados em regras ativas escritas em SQL – ART-TOOL (Active Rule Testing TOOL). Técnicas de teste estrutural de software foram estendidas para apoiar a aplicação de critérios de teste baseados em análise de fluxo de dados ao teste de regras ativas. A ferramenta foi usada para o teste de 15 regras ativas e os resultados são promissores: todos os defeitos foram revelados; e a ferramenta suportou a aplicação e a avaliação de cobertura dos critérios.*

1. Introdução

Aplicações de banco de dados estão presentes em muitas organizações e a busca de qualidade dessas aplicações é uma realidade em nosso dia-a-dia. Nesse contexto, uma das metas na construção de Sistemas de Informação baseados em bancos de dados é produzir software de alta qualidade, o qual pode ser alcançado pelo teste de software. Com a aplicação de um teste sistemático, é possível encontrar defeitos no software e, assim, melhorar a sua qualidade e a sua confiabilidade. Técnicas de teste são aplicadas em diversos tipos de programas, incluindo aqueles que fazem acesso a bases de dados.

[1] Este trabalho é parcialmente suportado pelo CNPq.

Sistemas de banco de dados tradicionais são passivos: a manipulação de dados só ocorre quando comandos são executados pelo sistema de banco de dados por uma requisição feita pelo usuário ou por programas de aplicação. Sistemas de bancos de dados ativos possuem regras ativas, as quais dão suporte a mecanismos que possibilitam a resposta automática a eventos, que ocorrem dentro ou fora do sistema, tais como, operações de manipulação de dados persistentes, eventos temporais, etc.

Regras ativas possuem muitas aplicações em Sistemas de Informação. Segundo Zaniolo *et al*. (1997), essas aplicações podem ser dividivas em duas classes: internas, que apóiam funções de gerenciamento de banco de dados, tais como manutenção de integridade, manutenção de dados e gerenciamento de replicações; e externas, freqüentemente chamadas de regras de negócio, que são executadas como parte do serviço da computação que normalmente estaria contido nos programas.

A linguagem SQL possui uso difundido na comunidade de bancos de dados (Fortier, 1999). Regras ativas escritas em SQL, também denominadas de *triggers*, têm sido usadas em aplicações de bancos de dados ativos, requerendo, portanto, abordagens sistemáticas de teste para a melhoria de sua qualidade.

O problema que motivou este trabalho foi a carência de técnicas de teste no contexto de regras ativas. O conjunto de regras ativas de um banco de dados é parte integrante do Sistema de Informação e, portanto, requer esforços na direção de melhoria de sua qualidade. Especificamente, as iniciativas de pesquisa existentes são focadas: no teste de regras escritas em linguagem declarativa e orientada a objetos (Chan *et al*., 1997); na cobertura de seqüências de regras (Vaduva, 1999); e na interação entre regras escritas em SQL (Leitão-Júnior, 2005), mas não exploram o teste individual de regras escritas em SQL.

A proposta deste trabalho é a automação do teste de Sistemas de Informação que utilizam regras ativas escritas em SQL, no contexto de teste estrutural de unidades. Representa o primeiro passo do teste de um conjunto de regras, o teste de cada regra separadamente, para posteriormente ser possível testar as interações entre as regras do conjunto. Foi construída uma ferramenta, denominada ART-TOOL – *Active Rule Testing Tool* – (Cardoso, 2004), escrita em linguagem Java, que apóia a aplicação de critérios de teste estrutural. Critérios de teste sistematizam a atividade de teste, estabelecendo requisitos que devem ser atendidos durante o teste. Critérios de teste estrutural derivam requisitos de teste a partir da estrutura interna do software. A ferramenta em sua versão corrente destina-se ao teste individual de regras, pela aplicação dos critérios estruturais *Potenciais Usos*; maiores esclarecimentos sobre tais critérios estão em (Maldonado, 1991), não sendo objetivo deste artigo uma revisão sobre critérios de teste e sobre os critérios *Potenciais Usos*.

A aplicação dos critérios *Potenciais Usos* apoiada pela ferramenta apresentou resultados promissores no teste de dois conjuntos de regras: a existência de todos os defeitos existentes foi revelada e foram viabilizadas a aplicação de casos de teste e a avaliação de cobertura dos critérios. Dessa forma, ganha-se confiança na corretude do software atribuído a Sistemas de Informação cujo teste foi apóiado pela ferramenta.

Este artigo está organizado da seguinte forma: na Seção 2 é abordado o teste de regras ativas, incluindo trabalhos existentes na área; a Seção 3 contém a descrição da ferramenta ART-TOOL; exemplo de aplicação e os resultados obtidos são apresentados

na Seção 4; um estudo de caso sobre o mecanismo de cursores é mostrado na Seção 5; na Seção 6 são apresentadas as conclusões do trabalho.

2. Teste de Regras Ativas – Revisão Bibliográfica

Os componentes de uma regra ativa – evento, condição e ação – são passíveis de conter defeitos. O evento descreve uma ocorrência à qual a regra deve ser capaz de responder; a condição examina o contexto no qual o evento ocorreu; e a ação descreve a tarefa a ser cumprida pela regra, se o evento ocorreu e a condição foi avaliada como verdadeira. Por exemplo, na ocorrência da venda de um produto (evento), se por isso o nível de estoque ficar abaixo do limite para este produto (condição), então um pedido de reposição de estoque deverá ser automaticamente conduzido (ação). Defeitos no evento, na condição e na ação da regra podem afetar: o disparo da regra, a decisão se a ação da regra deverá ser executada e as computações da tarefa da regra, respectivamente.

Os trabalhos que abordam teste de banco de dados em Sistemas de Informação enfatizam diferentes aspectos: geração automática de dados de teste (Lyon, 1977; Noble, 1983; Davies *et al*., 2000); teste de consultas (Mannila e Rih, 1989); teste de sistema de banco de dados (Roper e Rahim, 1993); teste de programa de aplicação (Chays *et al*., 2000; Spoto, 2000); teste do esquema da base de dados (Aranha *et al*., 2001).

No final da década de 90, surgiu o primeiro trabalho que explora o teste de regras ativas. Chan *et al*. (1997) adaptaram técnicas de teste de programas convencionais, procurando explorar o fluxo de controle intra- e inter-regra. A análise do fluxo de controle intra-regra depende de particularidades da especificação da linguagem da regra, que é a CDOL, uma linguagem orientada a objetos.

Vaduva (1999) busca determinar a existência de defeitos causados por conflitos e por dependências entre regras. O trabalho explora o fluxo de controle inter-regra e as interações entre regras, visando à detecção de comportamento defeituoso de regras. É utilizado o sistema SAMOS, um gerenciador de bancos de dados orientado a objetos.

A interação entre regras norteou o trabalho de Leitão-Júnior (2005). Foram propostos critérios de teste baseados na dependência de fluxo de controle e de fluxo de dados entre regras não necessariamente distintas. Diferentemente das abordagens de Chan *et al.* (1997) e Vaduva (1999), essa pesquisa explora a SQL e utiliza aspectos da estrutura interna das regras para derivar requisitos de teste.

3. A Ferramenta ART-TOOL

A ferramenta ART-TOOL (Cardoso, 2004) auxilia o testador no teste individual de regras ativas escritas em SQL. Conforme ressaltado na Seção 1, a ART-TOOL suporta a aplicação de um conjunto de critérios de teste, mais especificamente, a família de critérios *Potenciais Usos*: *Todos Potenciais Usos* e *Todos Potenciais Usos/du* (Maldonado, 1991). Em adição, outros critérios de teste são suportados, tais como *Todos Nós*, *Todos Arcos* e *Todos Usos*.

Nos critérios *Potenciais Usos*, é requerida a cobertura de associações que demandam a execução de caminhos de fluxo de controle entre a definição de uma variável (um valor é atribuído a uma posição de memória) e um potencial uso da definição da variável (um valor é lido da memória). No caso de regras ativas, tais

associações são abstraídas das ocorrências das variáveis locais à regra e das variáveis ligadas a dados persistentes. Dois trabalhos foram utilizados na construção da ferramenta. O primeiro refere-se a um suporte automatizado ao teste estrutural de programas convencionais, denominado POKE-TOOL (Chaim, 1991), alguns de seus módulos foram acoplados à ferramenta. O segundo inspirou a definição de modelos de implementação da ferramenta (Spoto, 2000).

A ART-TOOL foi projetada em módulos, como ilustra a Figura 1, compondo três etapas da atividade de teste: a análise estática, o teste da regra e a avaliação.

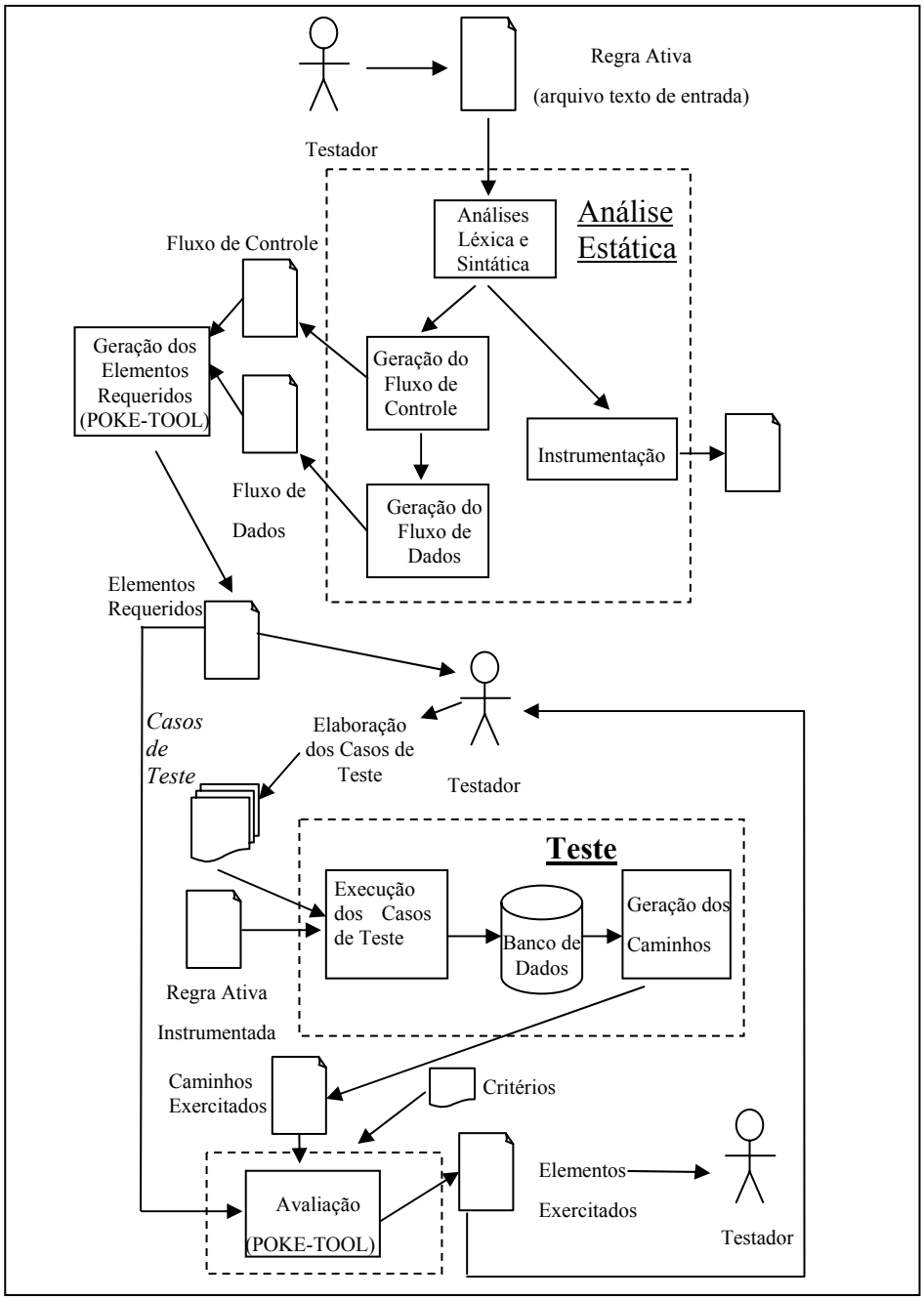

Figura 1. Arquitetura da ferramenta ART-TOOL.

Na análise estática é feita a preparação para o teste. O módulo de análises léxica e sintática lê o código da regra ativa e gera informações referentes aos comandos

analisados. Em seguida, o módulo de geração do fluxo de controle produz o grafo de fluxo de controle da regra ativa em teste. A partir desse grafo, pela execução do módulo de geração do fluxo de dados, as variáveis associadas a cada nó são analisadas, gerando o grafo de fluxo de dados. O módulo de instrumentação gera a nova versão da unidade a ser testada – a regra ativa com comandos de escrita inseridos – para tornar possível o acompanhamento da execução dos casos de teste. Finalizando a fase estática, o módulo de geração dos elementos requeridos provê um conjunto de caminhos e associações requeridos para satisfazer os critérios *Potenciais Usos*. Na etapa de teste, casos de teste são aplicados à versão instrumentada da regra, gerando resultados da execução da regra; especificamente, o módulo de geração dos caminhos exercitados produz os caminhos percorridos devido à aplicação de cada caso de teste. A avaliação verifica se um conjunto de casos de teste é adequado em relação ao critério aplicado. São determinados os caminhos (associações) efetivamente exercitados e verificado se o critério foi satisfeito; também é fornecida uma lista de caminhos requeridos e não exercitados pelo conjunto de casos de teste.

4. Aplicação da ART-TOOL

Com o objetivo de avaliar a aplicação dos critérios *Potencias Usos* no teste de regras ativas escritas em SQL, foram testadas quinze regras escritas para o sistema ORACLE, divididas em dois conjuntos. O primeiro é composto por regras elaboradas com o intuito de explorar os diversos comandos da SQL; o segundo constitui um conjunto real composto de cinco regras. Um aspecto motivador foi a aplicabilidade observada para os critérios *Potenciais Usos* e, conseqüentemente, para os critérios de teste menos exigentes (*Todos Nós*, *Todos Arcos*, *Todos Usos*). A ferramenta ART-TOOL tornou factível a aplicação dos casos de teste e forneceu suporte à avaliação dos critérios. Defeitos em comandos de acesso à base de dados e defeitos típicos de programas convencionais, dentre eles, os que envolvem variáveis locais, comandos de atribuição com variáveis locais, troca de variáveis locais, ausência de incremento e predicados incorretos, foram detectados nos dois conjuntos de regras testadas.

Esta seção ilustra a aplicação da ART-TOOL pelo teste de uma de regra ativa denominada R1, a qual está associada a um banco de dados baseado em Elmasri e Navathe (2003). O exemplo explora alguns comandos de manipulação da SQL e mostra como a ferramenta abstrai os elementos requeridos. A descrição da regra R1 e o seu código fonte são apresentados na Tabela 1. Nessa regra ativa, não existe o componente condição: o componente evento é especificado na cláusula CREATE TRIGGER e o componente ação é definido no bloco de comandos BEGIN-END abaixo da cláusula DECLARE. O início de cada linha do código fonte possui o padrão /* *n* */, onde *n* denota o número do nó correspondente no grafo de fluxo de controle da regra.

Tabela 1. Descrição resumida e código fonte da regra ativa R1

Regra R1 Descrição	Esta regra se inicia a partir de qualquer alteração, inclusão ou exclusão na tabela de empregados. Como nessa tabela é possível que apenas um empregado seja supervisor, a regra verifica a sua existência, analisa seu salário, que deve ser superior a todos os inclusos na tabela e o considera em uma tabela de ocorrências.
Código	/* *n* */ - Indica o número dos respectivos nós /*1*/ CREATE TRIGGER R1 AFTER DELETE OR INSERT OR UPDATE ON Employee DECLARE e_salaryerror EXCEPTION; v_qtde INTEGER; v_emplno employee.emplno%type; v_salary employee.salary%type; BEGIN /*2*/ SELECT emplno, salary INTO v_emplno, v_salary FROM Employee WHERE emplsupervisor IS NULL; /*3*/ INSERT INTO Log (timetamp, userid, status, description, tableid, tablekey, oldvalue, newvalue, alert) VALUES (SYSDATE, user, 'S', 'check president', 'employee', NULL, NULL, NULL, 'The salary of the president ' || TO_CHAR(v_emplno) || ' is ' || TO_CHAR(v_salary)); /*4*/ SELECT COUNT(*) INTO v_qtde FROM Employee WHERE (salary > v_salary); /*5*/ IF (v_qtde > 0) THEN /*6*/ RAISE e_salaryerror; /*7*/ END IF; /*8*/ EXCEPTION WHEN NO_DATA_FOUND THEN /*9*/ INSERT INTO Log (timetamp, userid, status, description, tableid, tablekey, oldvalue, newvalue, alert) VALUES (SYSDATE, user, 'E', 'check president', 'employee', NULL, NULL, NULL, 'The company has no president'); WHEN TOO_MANY_ROWS THEN /*10*/ INSERT INTO Log (timetamp, userid, status, description, tableid, tablekey, oldvalue, newvalue, alert) VALUES (SYSDATE, user, 'E', 'check president', 'employee', NULL, NULL, NULL, 'The company has too many presidents'); WHEN e_salaryerror THEN /*11*/ INSERT INTO Log (timetamp, userid, status, description, tableid, tablekey, oldvalue, newvalue, alert) VALUES (SYSDATE, user, 'E', 'check president', 'employee', NULL, NULL, NULL, 'This salary is not permited'); WHEN OTHERS THEN /*12*/ INSERT INTO Log (timetamp, userid, status, description, tableid, tablekey, oldvalue, newvalue, alert) VALUES (SYSDATE, user, 'E', 'check president', 'employee', NULL, NULL, NULL, 'Error checking president'); /*13*/ END;

O fluxo de controle da regra R1 é apresentado Figura 2. O nó 1 representa o evento da regra: a regra é disparada se as operações *delete*, *insert* ou *update* forem aplicadas à relação (tabela) *Employee*. Os nós 2, 3 e 4 representam comandos de manipulação da SQL; estes nós possuem dois arcos de saída: um para a execução normal do comando (linha contínua) e outro para a ocorrência de exceção na execução do comando (linha tracejada). Os nós 8 a 12 representam o bloco existente na regra para tratamento de exceção. O nó 13 é o nó de saída do grafo.

O fluxo de dados apresenta características específicas de regras ativas, conforme ilustrado na Tabela 2; estão descritos os nós, os arcos e as ocorrências de definição, de uso computacional (c-uso) e de uso predicativo (p-uso) das variáveis da regra R1. No nó de entrada do grafo (nó 1), ocorre a definição da variável persistente atribuída ao evento da regra (*Employee*) e das demais variáveis persistentes explicitamente referenciadas na regra (*Log*). O fluxo de dados devido às exceções que podem ocorrer nos comandos de manipulação da SQL – *insert*, *delete*, *update* e *select* – é mapeado para uma pseudovariável denominada *exception*. Nos nós desses comandos (nós 2, 3 e 4) ocorre a definição da variável *exception* e em seus arcos de saída (arcos (2,3), (2,8), (3,4), (3,8), (4,5) e (4,8)) tem-se o p-uso dessa variável. O p-uso da variável *exception* também é observado no bloco de tratamento de exceção (arcos (8,9), (8,10), e (8,12)), em uma estrutura de seleção múltipla para determinar o tipo de exceção ocorrida. *E_salaryerror* é uma variável declarada na regra, em que o programador explicitamente provoca uma exceção na execução da regra; esta variável é definida no nó atribuído ao comando *raise* (nó 6) e existe p-uso desta variável no arco de saída deste nó (arco (6,11)).

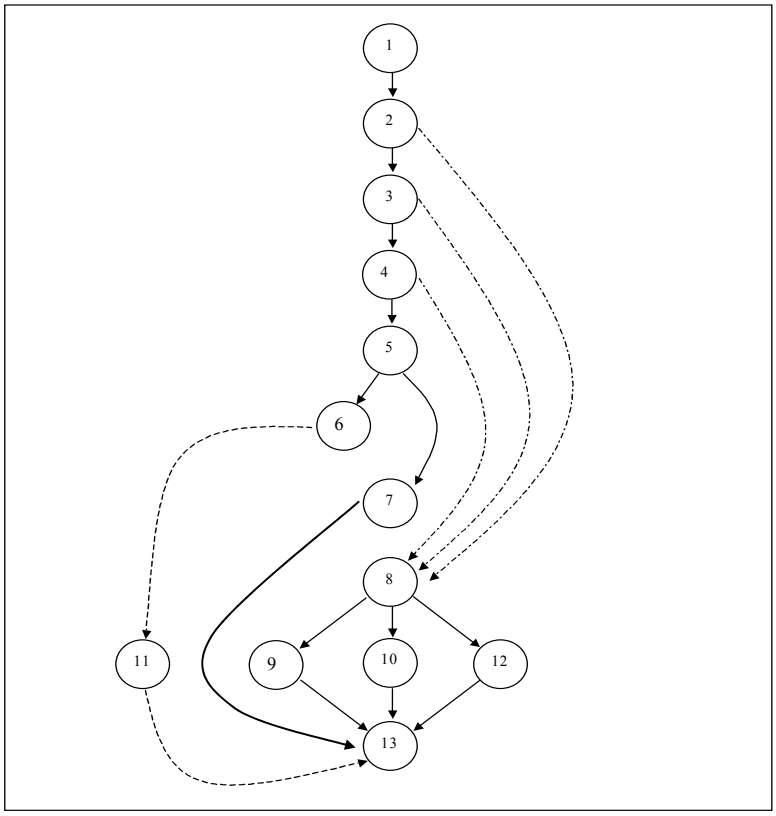

Figura 1. Representação do fluxo de controle da regra ativa R1

Tabela 2. Representação do fluxo de dados da regra ativa R1

Nó/ Arco	Definição	C-uso	P-uso
1	Employee, Log	Employee	
2	V_emplno, v_salary, exception	Employee	
3	Log, exception	v_emplno, v_salary	
4	V_qtde, exception	Employee, v_salary	
(2,3), (2,8), (3,4), (3,8), (4,5), (4,8)			exception
(5,6),(5,7)			v_qtde
6	E_salaryerror		
(8,9),(8,10), (8,12)			exception
9	Log		
10	Log		
11	Log		
(6, 11)			e_salaryerror
12	Log		

Os elementos requeridos da regra R1, para os critérios *Todos Nós*, *Todos Usos*, *Todos Potenciais Usos* e *Todos Potenciais Usos/du*, são mostrados na Tabela 3. Para o critério *Todos Nós*, são apresentados os nós que devem ser exercitados durante o teste (ou seja, todos os nós do grafo de fluxo de controle da regra); para os demais critérios, cada elemento requerido é apresentado pela tripla <*def, uso, var*>, onde: *def* é o nó em que ocorre definição de dados; *uso* é o nó ou arco em que ocorre uso de dados; e *var* refere-se à uma variável local à regra ou à uma variável persistente.

Tabela 3. Elementos requeridos da regra R1 para os critérios especificados

Critérios	**Elementos Requeridos**
Todos Nós	1 2 3 4 5 6 7 8 9 10 11 12 13
Todos Usos	<1,2, Employee>, <1,4, Employee>, <2,3, v_emplno>, <2,3, v_salary>, <2,4, v_salary> <2,(3,4), exception>, <2,(2,8), exception>, <4,(4,8), exception>, <4,(5,6), v_qtde> <2,(8,9), exception>, <2,(8,10), exception >, <2,(8,12), exception> <2,(3,8), exception>, <3,(3,4), exception>, <3,(4,5), exception>, <6,11, e_salaryerror> <3,(3,8), exception>, <3,(4,8), exception>, <4,(4,5), exception>, <4,(5,7), v_qtde>, <3,(8,9), exception>, <3,(8,10), exception >, <3,(8,12), exception > <4,(8,9), exception>, <4,(8,10), exception>, <4,(8,12), exception >
Todos Pot-Uso e Todos Pot-Uso/Du	<1,(2,8),{Employee,Log}>, <1,(3,8),{Employee}>, <1,(8,12),{Employee}>, <1,(8,12),{Log}>, <1,(8,10),{Employee}>, <1,(8,10),{Log}>, <1,(8,9),{Employee}>, <1,(8,9),{ Log }>, <1,(4,8),{Employee}>, <1,(5,7),{Employee}>, <1,(5,6),{Employee}>, <1,(2,3),{Employee,Log }> <2,(2,8),{v_emplno,v_salary,exception}>, <2,(3,8),{v_emplno,v_salary}>,<2,(8,12),{exception}> <2,(8,12),{v_emplno,v_salary}>,<2,(8,10),{exception}> <2,(8,10),{v_emplno,v_salary}>,<2,(8,9),{exception}> <2,(8,9),{v_emplno,v_salary}>,<2,(4,8),{v_emplno,v_salary}>, <2,(5,7),{v_emplno,v_salary}>, <2,(2,3),{v_emplno,v_salary,exception}>, <2,(5,6),{v_emplno,v_salary}>, <3,(3,8),{ Log, exception }>, <3,(8,12),{Log}>, <3,(8,12),{exception}>, <3,(8,10),{ Log }>, <3,(8,10),{exception}>, <3,(8,9),{ Log }>, <3,(8,9),{ exception }>, <3,(4,8),{ Log }>, <3,(5,7),{ Log }>, <3,(6,11),{ Log }>, <3,(5,6),{ Log }>, <3,(3,4),{ Log, exception }> <4,(8,12),{ exception }>, <4,(8,12),{ v_qtde }>, <4,(8,10),{ exception }>, <4,(8,10),{ v_qtde }>, <4,(8,9),{ exception }>, <4,(8,9),{ v_qtde }>, <4,(4,8),{ v_qtde, exception }>, <4,(5,7),{ v_qtde, exception }>, <4,(5,6),{ v_qtde, exception }> <6,{e_salaryerror}>, <9,{Log}>, <10,{Log}>, <11,{Log}>, <12,{Log}>

A Tabela 4 ilustra os elementos exercitados para os critérios *Todos Nós*, *Todos Usos*, *Todos Potenciais Usos* e *Todos Potencias Usos/du* pela aplicação de um conjunto de casos de teste. Comparando os conteúdos das Tabelas 3 e 4, pode-se concluir que, neste conjunto de casos de teste, não houve a cobertura completa dos elementos requeridos pelos critérios; por exemplo, para o critérios *Todos Nós*, foram exercitados apenas os *Nós* 1, 2, 8, 9 e 13, apesar do exercício de todos os nós ter sido requerido.

Tabela 4. Elementos exercitados na aplicação dos critérios selecionados

Descrição do caso de teste – C1	Atualização do salário de um supervisor e inclusão de seu supervisor. Situação não permitida na tabela EMPLOYEE, tabela sem supervisor; visa o exercício de uma exceção.
Critérios	**Elementos Exercitados**
Todos Nós	1 2 8 9 13
Todos Usos	<1,2, Employee>, <2,(2,8), exception>, <2,(8,9), exception>
Todos Pot-Usos	<1,(2,8),{Employee, Log}>, <1,(8,9),{ Employee }>, <1,(8,9),{ Log }> <2,(2,8),{v_emplno,v_salary, exception}> ,<2,(8,9),{exception}> <2,(8,9),{v_emplno, v_salary}>, <9,(,),{ Log }>
Todos Pot-Uso/Du	<1,(2,8),{Employee, Log}>, <1,(8,9),{ Employee }>, <1,(8,9),{ Log }> <2,(2,8),{v_emplno,v_salary, exception}>, <2,(8,9),{ exception }> <2,(8,9),{v_emplno,v_salary}>, <9,(,),{ Log }>

A Tabela 5 exibe a cobertura atingida para todos os critérios avaliados pela aplicação dos conjuntos de casos de teste 1, 2 3 e 4. Por ser um critério mais exigente, o critério *Todos Potenciais DU-Caminhos* apresentou uma cobertura bem inferior em relação aos demais critérios; o oposto pode ser afirmado com respeito ao critérios *Todos Nós*. A Tabela 5 ilustra o potencial da ART-TOOL com respeito à avaliação da aplicação de critérios de teste em Sistemas de Informação baseados em regras ativas escritas em SQL.

Tabela 5. Cobertura dos critérios avaliados

	Critérios							
Casos de Teste– R1	Todos Nós	Todos Arcos	Todos Usos	Todos P-Usos	Todos Pot-Usos	Pot-Du Caminhos	Pot-Usos /Du	Du Cami-nhos
1	38 %	25 %	11 %	10 %	14 %	8 %	14 %	12 %
2	38 %	25 %	11 %	10 %	14 %	8 %	14 %	12 %
1 e 2	46 %	38 %	15 %	14 %	24 %	15 %	24 %	16 %
3	54 %	13 %	41 %	29 %	14 %	10 %	14 %	36 %
1, 2 e 3	77 %	50 %	52 %	43 %	38 %	25 %	38 %	48 %
4	62 %	13 %	44 %	29 %	20 %	15 %	20 %	40 %
1, 2, 3 e 4	92 %	63 %	59 %	48 %	52 %	40 %	52 %	56 %

5. Estudo de Caso – Uso de Cursores

O mecanismo de cursor é muito utilizado em Sistemas de Informação com programação SQL, pois permite o manuseio *tupla-a-tupla* (linha-a-linha) dos elementos de uma relação (tabela) da base de dados. Cursores estão associados a comandos de consulta a bases de dados, habilitando o acesso a cada *tupla* do resultado da execução da consulta. Esta seção exemplifica como ocorre a descoberta de defeitos pelo uso da ART-TOOL.

A cláusula DECLARE CURSOR declara uma variável que identifica o comando de consulta atribuído ao cursor, denominada *variável cursor*. A execução do comando OPEN inicia a consulta, permitindo que o seu resultado seja acessado pelo comando FETCH. A execução do comando FETCH acessa a próxima *tupla* do resultado da consulta. O comando CLOSE encerra o uso do cursor. Uma utilização típica do mecanismo de cursor é a execução do comando OPEN, seguida por execuções iterativas do comando FETCH e, por fim, pela execução do comando CLOSE.

A partir das diretrizes dos modelos de implementação da ART-TOOL, são abstraídas associações de fluxo de dados que requerem a execução da seqüência de pares de comandos OPEN-FETCH, FETCH-CLOSE, OPEN-CLOSE, FETCH-FETCH, OPEN-OPEN, CLOSE-CLOSE e CLOSE-FETCH. A três últimas seqüências representam cenários de defeito e, portanto, deveriam idealmente estar ligadas a associações de fluxo de dados que não podem ser cobertas (não exercitáveis). A Figura 3 ilustra cenários para as seqüências CLOSE-CLOSE (caso 1) e OPEN-OPEN (caso 2). Associações de fluxo de dados são requeridas pelos critérios Potenciais Usos para o exercício desses cenários, aumentando as chances de manifestação de falhas.

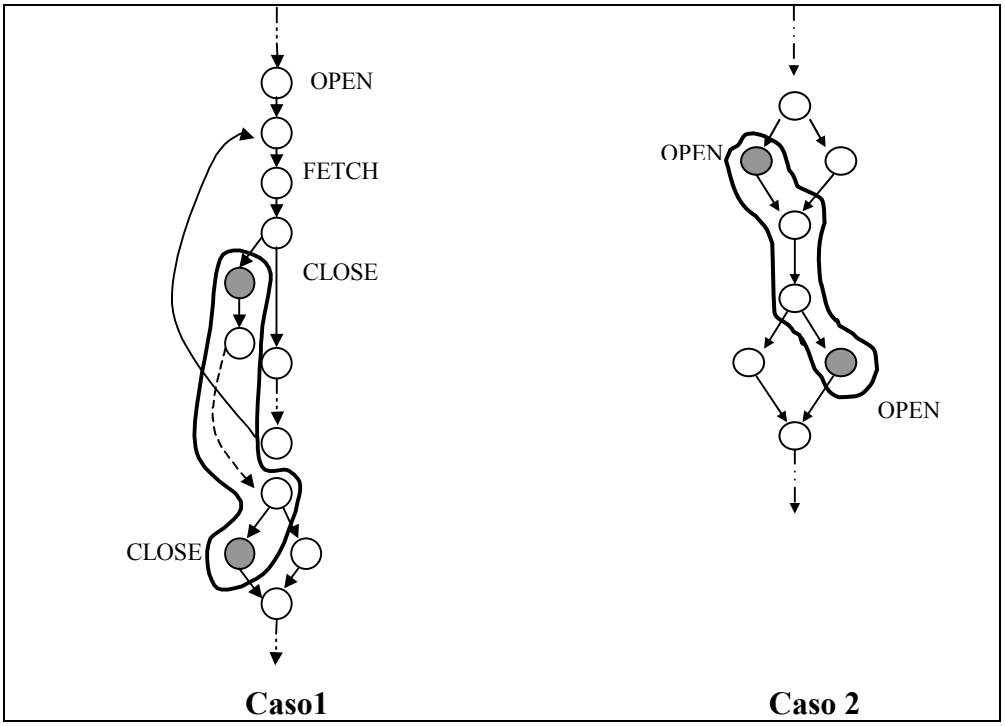

Figura 2. Exemplo de cenários de defeito para o uso de cursores

6. Conclusão

A proposta deste trabalho é contribuir para a melhoria de qualidade de Sistemas de Informação em aplicações de bancos de dados, pela automação do teste de regras ativas escritas em SQL. Foi construído um suporte automatizado denominado ART-TOOL (*Active Rule Testing TOOL*) (Cardoso, 2004), escrito na linguagem Java, que apóia a aplicação de critérios estruturais baseados em fluxo de dados, especificamente os critérios Potenciais Usos (Maldonado, 1991).

Os resultados do teste de quinze regras ativas são promissores: a aplicação dos critérios Potenciais Usos apóia a descoberta de defeitos em regras ativas escritas em SQL; e a ferramenta ART-TOOL tornou factível a aplicação dos casos de teste e forneceu suporte à avaliação de cobertura dos critérios. Defeitos em comandos de acesso à base de dados e defeitos típicos de programas convencionais, dentre eles, os que envolvem variáveis locais, comandos de atribuição com variáveis locais, troca de variáveis locais, ausência de incremento, predicados incorretos, etc., foram detectados nas regras. Os resultados demonstram a contribuição deste trabalho na busca de qualidade em Sistemas de Informação baseados em regras ativas escritas em SQL. Representa, também, um valioso recurso ao desenvolvedor de Sistemas de Informação, pois fornece maior confiança na corretude das regras ativas.

Algumas evoluções para este trabalho são: (i) utilizar análises de fluxo de dados persistentes mais precisas, ensaiando empiricamente qual o impacto no custo e na descoberta de defeitos: significa tornar a aplicação dos critérios mais exigente, pois a cobertura de associações dar-se-ía em níveis mais rigorosos: a definição e o uso ocorreriam na mesma *tupla* e atributo da relação; (ii) estender a aplicação dos critérios Potenciais Usos à interação entre regras e comparar os resultados com os obtidos por Leitão-Júnior (2005); (iii) estender as análises às regras suportadas por vários sistemas gerenciadores de banco de dados.

Referências

Aranha, M.C.L.F.M., Mendes, N.C., Jino, M. e Toledo, C.M.T. (2001) "RDBTool: Uma ferramenta de Apoio ao Teste de Bases de Dados Relacionais", Anais do XI CITS: Conferência Internacional de Tecnologia de Software: Qualidade de Software, Cutitiba, PR., Brasil, p. 31-43.

Cardoso, V. M. (2004) "Uma Ferramenta para Teste Estrutural de Regras Ativas em Linguagem SQL", Dissertação de mestrado, Faculdade de Engenharia Elétrica e de Computação , Unicamp, Campinas, S.P., Brasil.

Chaim, M.L. (1991) "POKE-TOOL – Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados", Dissertação de mestrado, Faculdade de Engenharia Elétrica, Unicamp, Campinas, S.P., Brasil.

Chan, H.W.R., Dietrich, S.W. e Urban, S.U. (1997) "On Control Flow Testing of Active Rules in a Declarative Object-Oriented Framework", Proceedings of the Third International Workshop on Rules in Database Systems (RIDS '97), Skoevde, Suécia.

Chays, D., Dan, S., Frankl, P.G., Vokolos, F.I. e Weyuker, E.J. (2000) "A Framework for Testing Database Applications", Proceedings of the International Symposium on Software Testing and Analysis, Portland, Oregon.

Davies, R.A., Beynon, R.J.A. e Jones, B.F. (2000) "Automating the Testing of Databases", Proceedings of the First International Workshop on Automated Program Analysis, Testing and Verification, University of Limerick, Irlanda.

Elmasri, R, Navathe, S.B. (2003), Fundamentals of Database System, Addison-Wesley., 3ª edição.

Fortier, P.J. (1999), SQL-3 Implementing the Object-Relational Database, McGraw-Hill Enterprise Computing Series.

Leitão-Júnior, P. S. (2005) "Teste Baseado na Interação entre Regras Ativas Escritas em SQL", Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, Campinas, S.P., Brasil.

Lyon, N. R. (1977) "An Automatic Data Generating System for Data Base Simulation and Testing", Database, vol. 8, nº 4, p.10-13.

Maldonado, J.C. (1991) "Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software", Tese de doutorado, Faculdade de Engenharia Elétrica, Unicamp, Campinas, S.P., Brasil.

Mannila, H. e Raiha, K.J. (1989) "Automatic Generation of Test Data for Relational Queries", Journal of Computer and System Sciences, vol.38, nº 2, p. 240-258.

Noble, H. (1983) "The Automatic Generation of Test Data for a Relational Database", Information Systems, vol. 8, nº 2, p. 79-83.

Roper, M. e Rahim, A. R. B. A. (1993) "Software Testing Using Analysis and Design Based Techniques". Software Testing, Verification and Reliability, vol. 3, p. 165-179.

Spoto, E.S. (2000) "Critérios de Teste Estrutural para Programas de Aplicação de Banco de Dados Relacional", Tese de doutorado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, Campinas, S.P., Brasil.

Vaduva, A., (1999) "Rule Development for Active Database Systems", Tese de doutorado, Faculdade de Economia, Universidade de Zurique, Zurique.

Zaniolo, C., Ceri, S., Faloutsos, C., Snoograss, R. T., Subrahmanian, V. S. e Zicari, R. (1997) "Advanced Database Systems". Morgan Kaufmann Publishers, Inc. San Francisco, California, USA.