Teste Estrutural de Integração de Programas de Aplicação de Banco de Dados Relacional

Edmundo Sérgio Spoto[1], Plínio de Sá Leitão Junior[2] , Mario Jino[2] e José Carlos Maldonado[3]

[1]UNIVEM/FEESR - Av. Hygino Muzzi Filho, 529 – Cmps Univ. Cep: 17.525-901, Cx. Postal 2041, Marília – SP, Brasil – (dino@fundanet.br)

[2]DCA/FEEC/UNICAMP - Av. Albert Einstein, 400, Cep: 13083-970, Cx. Postal 6101, Campinas - SP, Brasil (plinio@dca.fee.unicamp.br; jino@dca.fee.unicamp.br)

[3]ICMC-USP - Av. Dr. Carlos Botelho, 1465, - Cep: 13560-970, Cx. Postal 668, São Carlos - SP, Brasil (jcmaldon@icmc.usp.br)

***Abstrac**t. The search for quality in Relational Database Applications (RDA) involves the definition of testing approaches that consider persistent data flow. The structural testing technique is based on the internal structure of programs; it is applied in the unit testing of programs and in the integration testing of those units. This work explores the analysis of persistent dataflow in RDA, to determine requirements for structural integration testing. Specifically, an approach is proposed for the structural integration testing of programs with embedded SQL, where the persistence of data establishes data flow associations. The approach considers intra-modular and inter-modular integration testing to define test requirements. The testing criteria proposed are based on the call structure of units and on the exclusive data dependence, focused on the relationships of persistent dataflow.*

***Resumo.** A busca pela qualidade de Aplicações de Banco de Dados Relacionais (ABDR) envolve a definição de abordagens de teste que considerem o fluxo de dados persistentes. A técnica de teste estrutural é baseada na estrutura interna de programas; ela é aplicada no teste de unidade de programas e na integração dessas unidades. Este trabalho explora a análise de fluxo de dados persistentes em ABDR, para determinar requisitos para teste estrutural de integração. Especificamente, uma abordagem é proposta para o teste estrutural de integração em programas com SQL embutida, onde a persistência de dados estabelece associações de fluxo de dados. A abordagem considera o teste de integração intra-modular e inter-modular para definir os requisitos de teste. Os critérios de teste propostos estão baseados na estrutura de chamada entre unidades e na dependência exclusiva de dados, focados nas relações de fluxo de dados persistentes.*

1. Introdução

A manipulação de dados persistentes desempenha um importante papel em soluções implementadas por software. A persistência de dados é um atributo ligado à necessidade por dados não voláteis na execução de aplicações. Aplicações que manipulam dados persistentes são distintas das aplicações convencionais, pois incorporam tais dados ao espaço de entrada e de saída durante a sua execução.

O uso de bancos de dados relacionais tem-se difundido em aplicações que manipulam dados persistentes. Tais aplicações, denominadas Aplicações de Banco de

Dados Relacional (ABDR), tipicamente recuperam e ou atualizam as relações das bases de dados. Essas relações representam os dados persistentes que, em conjunto com as variáveis declaradas no programa, compõem as variáveis de uma aplicação.

Similares a programas convencionais, ABDRs requerem esforços para o controle e para a melhoria de sua qualidade. Teste de software é uma atividade usada para dar evidências da confiabilidade e da qualidade de software. Assim, é pertinente a busca por abordagens voltadas ao teste de ABDRs. Especificamente, a técnica de teste estrutural é baseada na estrutura interna de programas, focando o fluxo de controle e o fluxo de dados. O teste se divide em 2 estágios: teste de unidade de programa – menor parte executável de um programa, como uma função ou procedimento – e teste de integração dessas unidades. O foco deste artigo é a aplicação da técnica de teste estrutural no estágio de integração, no contexto de aplicações que manipulam bancos de dados relacionais.

A geração de bases de dados é uma área pertinente ao teste de aplicações que manipulam dados persistentes. A utilização de bases dedicadas à atividade de teste usualmente alcança uma reduzida amostra dos dados, a qual pode não ser representativa dos cenários reais de uso, especialmente em termos de teste de performance. Por outro lado, o uso de bases reais pode potencialmente expor dados protegidos contra acessos não autorizados. Wu et al. investigam técnicas para a aplicação de bases de produção ao teste de aplicações, sem revelar qualquer informação confidencial presente nessas bases [WU04]; a idéia é simular ambientes reais para propósitos de teste, mantendo a privacidade dos dados na base original. Chays et al. também estudam a geração de bases de teste, apresentando uma abordagem baseada na indicação do testador de dados típicos para cada atributo da base de dados [CHA00]; em [CHA03] é apresentada uma ferramenta que automatiza esta abordagem, auxiliando a aplicação de casos de teste e a checagem se as saídas e o novo estado da base de dados estão consistentes com o comportamento esperado.

Um aspecto pertinente é a determinação de requisitos que deveriam ser cumpridos durante o teste, visando a aumentar a propensão para revelar os defeitos existentes no software. Esses requisitos são comumente definidos por critérios de teste e estão associados a modelos. No caso do teste estrutural, o *modelo de fluxo de controle* é usado para definir critérios que exploram a transferência de fluxo de controle da aplicação. Para exemplificar, considere o grafo de *fluxo de controle* (GFC) de um programa *P*, que é um grafo dirigido, denotado por $G(P) = (N, A, e, s)$: N é o conjunto de nós, onde cada nó é uma seqüência de comandos executados sem transferência de controle, também denominado *bloco*; A é o conjunto de arcos representando a transferência de controle entre nós; e é o nó de entrada; e s é o nó de saída [CLA89, MAL91]. O critério *todos os nós* requer a execução de todos os nós durante o teste; o critério *todos os arcos* requer o exercício de todas as transferências de controle.

Por outro lado, o critério *todos os caminhos* explora as noções de *caminho* e de *caminho completo*: um *caminho* é uma seqüência finita de nós $(n_1, n_2, ..., n_k)$, $k \geq 2$, tal que, para todo nó n_i, $1 \geq i \geq k-1$, existe um arco (n_i, n_{i+1}) para $i = 1, 2, ...k-1$; um *caminho completo* é um caminho cujos nós inicial e final são, respectivamente, o nó de entrada e o nó de saída. O critério *todos os caminho*s requer que todos os caminhos completos sejam exercitados durante o teste; em geral, esse critério é impraticável, visto que a quantidade de caminhos completos é elevada, sobretudo quando existem laços. Os elementos exigidos por um critério de teste são denominados *elementos requeridos*.

Um outro modelo, denominado *modelo de fluxo de dados*, representa as ocorrências de definição e de uso de variáveis de programa. Uma *definição de variável*

ocorre quando um valor for armazenado em uma posição de memória. Um *uso de variável* ocorre quando houver uma recuperação de valor de uma posição de memória associada à variável, sem que se esteja definindo a variável. Os critérios de teste baseados no fluxo de dados determinam requisitos de teste para estabelecer associações entre a definição e o uso de variáveis – *associações def-uso*; por exemplo, se uma variável é definida em um nó do programa, é necessário exercitar caminhos entre essa definição e (potenciais) usos dessa variável. É intuitivo que se um valor for atribuído a uma variável, esse valor deverá ser usado no programa. Tais critérios são em geral mais exigentes do que os critérios *todos os nós* e *todos os arcos*, pois demandam testes mais rigorosos, e menos exigentes do que o critério *todos os caminhos*. Os critérios propostos por Rapps e Weyuker exigem a ocorrência explícita de uso de variável nos elementos requeridos [RAP82, RAP85]; a proposta de Maldonado et al. não exigem a ocorrência explícita de uso de variável e usa a idéia de potencial uso [MAL88, MAL91]. Em ABDR, a análise de fluxo de dados deve contemplar as variáveis persistentes pois essas variáveis estabelecem associações adicionais às associações de programas convencionais. Critérios específicos foram propostos em [SPO00a], visando a exercitar o teste de ABDR para *variáveis tabelas* presentes nesses programas. As variáveis tabelas possuem um enfoque mais amplo por serem variáveis persistentes cuja definição só é concretizada quando for validada a transação da base de dados (utilizada pelo comando COMMIT) [SPO00b].

Explorar relações de fluxo de dados persistentes em ABDR envolve aspectos importantes como: definir um modelo de fluxo de controle que inclua os comandos de acesso à base de dados e às estruturas de tratamento de exceção típicas nessas aplicações; analisar as ocorrências de definição e de uso de dados persistentes; especificar como as relações de fluxo de dados são caracterizadas entre as unidades de programa e entre programas; definir um modelo que contemple as relações de fluxo de dados; estabelecer requisitos de teste com base nos modelos de fluxo de controle e de fluxo de dados.

O problema que norteia este trabalho é a carência de critérios de teste estrutural de integração para ABDR, envolvendo o uso da **SQL** (*structured query language*) para a manipulação de dados persistentes; programas com essa característica são chamados de programas com **SQL** embutida. Os critérios baseados em fluxo de dados para programas convencionais são inadequados para variáveis persistentes, pois estas possuem particularidades como: referência à mesma *tupla* para caracterizar uma *associação def-uso*, aspectos de consistência transacional que requerem a confirmação (*commit*) ou o cancelamento (*rollback*) de transações e o relacionamento entre as relações da base de dados. Leitão et al. discutem questões relativas a defeitos em comandos SQL que manipulam dados persistentes, estudam como falhas podem ser manifestadas e constroem um mapeamento entre tipo de defeitos e dimensões de falhas visando a abstrair os fatores que influenciam a propagação de defeitos até a saída de execução de comandos [LEI05].

O artigo explora o teste de integração, que é executado após o teste de unidades de programa, e representa uma técnica sistemática para construir a estrutura do software e para revelar defeitos associados à interação entre unidades. Requisitos para o teste de integração são estudados, baseados na análise de fluxo de dados persistentes. Basicamente, trabalha-se com a noção de *módulo*, que neste artigo representa um programa de aplicação. São sugeridos requisitos de teste de integração *intra-modular* e

inter-modular, inspirados, respectivamente, em *associações def-uso* entre unidades de um programa e entre programas.

Os critérios de teste propostos baseiam-se em relações de fluxo de dados devido à dependência propriamente dita de dados persistentes ocasionadas nas definições que uma unidade U_A ocasiona na tabela da base de dados e seu respectivo uso ocasionado por outra unidade U_B.

Introduzem-se na Seção 1 o contexto, o problema e a questão investigada no trabalho; na Seção 2, são estudados os aspectos de fluxo de controle e de fluxo de dados abstraídos para ABDRs com SQL embutida, onde se ressalta a dependência de dados devido à persistência de dados; o teste estrutural de integração para ABDRs, baseado na dependência de dados persistentes, é explorado na Seção 3; na Seção 4 introduzimos os critérios de teste de integração propostos, classificando-os como intra-modular e inter-modular, e apresentamos um exemplo de seus ; as conclusões são apresentadas na Seção 5.

2. Definições Básicas e Terminologia

O teste estrutural de programas de ABDR com SQL embutida possui características similares às do teste de programas convencionais. As técnicas de teste definidas para programas convencionais podem ser aplicadas em ABDRs a partir de algumas adaptações, tendo em vista a presença de comandos SQL, variáveis tabela e variáveis host, não existentes em programas convencionais. Os conceitos básicos apresentados nesta seção são os relativos aos critérios estruturais de teste de programas convencionais, modificados e estendidos para os critérios estruturais de teste utilizados em ABDR. Na elaboração da abordagem de teste foi necessário: i) adaptar os conceitos e critérios de testes de unidade e de integração de programas convencionais para programas de ABDR; e ii) criar novos critérios de teste estrutural específicos para o escopo de banco de dados relacional (BDR).

Uma aplicação de banco de dados relacional é composta por um ou mais *Módulos*; cada *Módulo*[1] é composto por um conjunto de *Unidades de Programa* (*UP*), que são os procedimentos e funções do programa; um Módulo Mod_i pode ser decomposto em várias *UP*s, onde $1 \leq i \leq n$. Os Módulos geralmente podem ser codificados em diferentes linguagens, denominadas linguagens hospedeiras com **SQL** embutida. Cada *UP* é uma seqüência finita de comandos da linguagem hospedeira e comandos da linguagem **SQL**. A base de dados é representada pelas tabelas Tab_j onde $1 \leq j \leq m$. A Figura 1 apresenta a idéia geral de uma ABDR composta por: módulos Mod_1, Mod_2,.., Mod_n, com suas respectivas unidades *UPs*, e as tabelas de Tab_1, Tab_2, ..., Tab_m.

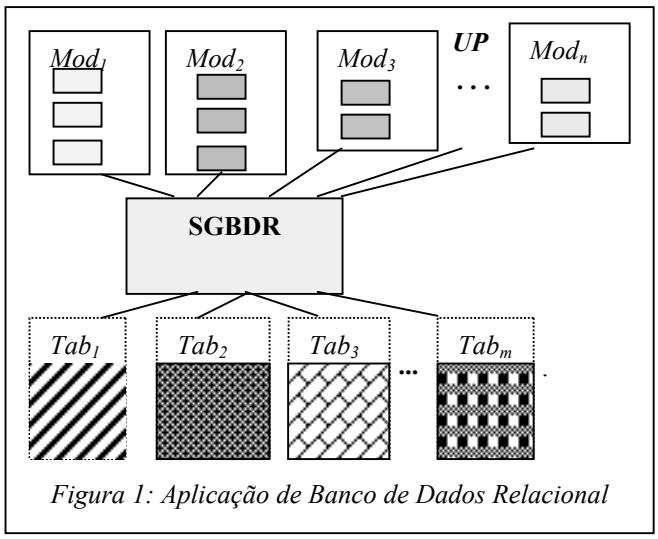

Figura 1: Aplicação de Banco de Dados Relacional

Similarmente a programas convencionais, a estrutura de controle de programas de aplicação também pode ser representada por um grafo de fluxo de

[2] A palavra **Módulo** usada nesta abordagem corresponde a um programa de ABDR.

controle. Entretanto, devido à existência dos comandos **SQL**, a representação gráfica de uma unidade de programa é um grafo de fluxo de controle estendido para ABDR com uso de **SQL**. A representação de grafo de programa acomoda os comandos executáveis da **SQL** em nós especiais, um em cada nó. Como notação gráfica, adotaram-se nós circulares para representar os blocos de comandos da linguagem hospedeira e de comandos declarativos da **SQL,** e nós retangulares para representar os comandos executáveis da **SQL:** INSERT, DELETE, UPDATE, SELECT, COMMIT, ROLLBACK, entre outros; um exemplo de GFC é exibido na Figura 2.

2.1 Fluxo de Controle de Programa de ABDR

O grafo de programa para representar uma UP de uma ABDR é denotado por G(UP) = (N^{BD}, E, n_{in}, n_{out}) onde: $N^{BD} = N_h \cup N_S$, N_h é o conjunto de nós provenientes da linguagem hospedeira e N_S é o conjunto de nós provenientes da linguagem **SQL**; E é o conjunto de arcos onde $E \subseteq N^{BD} \times N^{BD}$; o nó $n_{in} \in N_h$ é o nó de entrada e $n_{out} \in N^{BD}$ é o nó de saída do grafo de programa. O grafo de programa estabelece uma correspondência entre os comandos do programa representados pelos conjuntos de nós N_h e N_S, indicando os possíveis fluxos de controle entre os nós através dos arcos [SPO00b]. Os conjuntos $Arc_{in}(i)$ e $Arc_{out}(i)$ representam os conjuntos de arcos que, respectivamente, chegam e saem do nó i. A Figura 2 mostra o grafo de programa para a Unidade *InsFin*, que é uma unidade de programa extraída de [SPO00b], cuja linguagem hospedeira é a Linguagem C++.

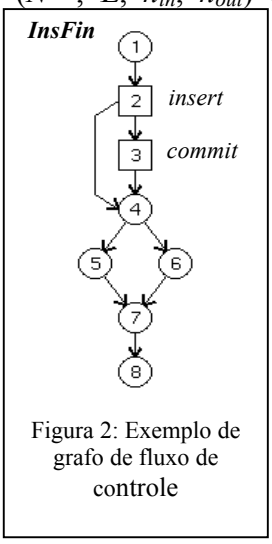

Figura 2: Exemplo de grafo de fluxo de controle

Para o tratamento de exceção foi utilizado o comando declarativo da **SQL** *"EXEC SQL WHENEVER SQLERROR GOTO end"*; no exemplo da Figura 2, este comando está no nó 1 (junto com os comandos da Linguagem C++) e ocasiona a existência do arco *(2,4)* com a condição de um desvio condicional para o rótulo *"end"* presente no nó *4*.

2.2 Fluxo de Dados de Programa de ABDR

A análise do fluxo de dados baseia-se nos tipos de ocorrência das variáveis de um programa de aplicação. Neste caso, podem existir três tipos de variáveis: i) variáveis de programa: são as variáveis manipuláveis somente por comandos da linguagem hospedeira; ii) variáveis host: são as variáveis que estabelecem os fluxos de dados entre a base de dados e o programa; e iii) variáveis tabela: são as variáveis tabela básicas (variáveis persistentes que compõem a base de dados) e as variáveis tabela de visão (tabelas virtuais derivadas das tabelas básicas e tratadas como variáveis locais ao Módulo de Programa). Cada tipo de variável tem o seu escopo de validade. Uma variável de programa é processada apenas no âmbito da linguagem hospedeira, uma variável host é processada tanto na linguagem hospedeira como na linguagem SQL e uma variável tabela é processada apenas na linguagem SQL e não precisa ser declarada no escopo do Módulo de Programa.

As *variáveis host* podem ser definidas e usadas em qualquer parte do programa (na linguagem hospedeira ou na linguagem **SQL**); denominamos de *s-uso* o uso de **SQL** relacionado às *variáveis host* e às *variáveis tabelas* que são usadas nos comandos da **SQL**. Para distinguir do uso na memória interna ocasionada pelas *variáveis host*, as

variáveis tabela são definidas e usadas em memórias secundárias. Neste caso denominaremos de *t-uso* o uso da *variável persistente* ou *variável tabela* [SPO00b].

A abordagem adotada de teste estrutural para programas de ABDR considera dois tipos de fluxos de dados para as variáveis tabela: *i)* fluxo de dados *intra-modular* (aborda o teste de unidade e teste de integração *intra-módular*); e *ii)* fluxo de dados *inter-modular*; em ambos os casos, a análise de fluxo de dados é usada para abstrair requisitos para o teste de integração *intra-modular* e *inter-modular*. O fluxo de dados *intra-modular* é observado entre as *UPs* de um mesmo módulo; o fluxo de dados *inter-modular* ocorre entre *UPs* de módulos distintos.

O teste de integração é aplicado depois que todas as unidades que compõem um programa foram devidamente testadas, isoladamente (*teste de unidade*). Para o teste de unidade em programas de aplicação, existe uma diversidade de critérios de teste estrutural, baseado em fluxo de controle e em fluxo de dados [RAP85, MAL91] que podem ser aplicados [SPO00a].

A dependência de dados é um aspecto fundamental associado à interação entre Unidades de Programas e Módulos da Aplicação. As dependências de dados existentes entre as *UPs* de programas convencionais são ocasionadas pelas variáveis globais ou por variáveis passadas por parâmetros por comandos de chamadas. ABDRs possuem dependências de dados baseadas nos comandos de chamada, como acontece nos programas convencionais e dependências de dados baseadas nas tabelas da base de dados. As dependências de dados entre as *UPs* e os Módulos de Aplicação de uma ABDR dão ensejo a novas abordagens de integração, propostas neste artigo.

Para apresentar os conceitos de teste de integração utilizados em programas de ABDR, foram apresentadas algumas definições importantes para o entendimento deste artigo. Um *c-uso* afeta diretamente uma computação que está sendo realizada, ou permite observar o valor de uma variável definida anteriormente. Um *p-uso* afeta diretamente o fluxo de controle do programa. Um caminho é *simples* se todos os nós que o compõem, exceto possivelmente o primeiro e o último, são distintos. Se todos os nós são distintos, diz-se que o caminho é um *caminho livre de laço*. Um caminho $(i, n_1, ..., n_m, j)$, $m \geq 0$, com uma definição da variável x no nó i e que não contenha redefinição de x nos nós $n_1, ..., n_m$ é chamado de *caminho livre de definição* com respeito a (c.r.a) x do nó i ao nó j e do nó i ao arco (n_m, j). Um nó n_i possui uma *definição global* de uma variável x se ocorrer uma definição de x no nó n_i e existir um caminho livre de definição de n_i para algum nó ou para algum arco que possua, respectivamente, um *c-uso* ou um *p-uso* de x. Um caminho $(n_1, n_2, ..., n_k)$ é um *du-caminho* c.r.a variável x se n_1 tiver uma definição global de x e: (1) ou n_k tem um *c-uso* de x e $(n_1, n_2, ..., n_j, n_k)$ é um é um caminho simples livre de definição c.r.a x; ou (2) (n_j, n_k) tem um *p-uso* de x e $(n_1, n_2, ..., n_j, n_k)$ é um caminho livre de definição c.r.a x e $n_1, n_2, ..., n_j$ é um caminho livre de laço [MAL91].

Os conjuntos utilizados para descrever os conceitos básicos dos critérios de testes para programas de aplicação são:

(i) *s-uso*(i)={variáveis com *s*-uso no nó $n_i \in N_s$};

(ii) $defT<n_\alpha,n_\beta>$ = {Variável ***t*** | ***t*** possui uma *definição persistente* pela concatenação dos nós n_α e n_β, representado por $< n_\alpha, n_\beta>$ onde n_α e $n_\beta \in N_s$}; o nó n_α refere-se ao comando INSERT, UPDATE ou DELETE; o nó n_β refere-se à confirmação da transação, pela ocorrência explícita ou implícita do comando COMMIT; a definição persistente é caracterizada quando o caminho executado possuir os nós n_α e n_β;

(iii) dsu(v,i)={nós $n_j \in N_s$ | $\boldsymbol{v} \in$ *s-uso*(n_j), existe um caminho livre de definição c.r.a. $\boldsymbol{v}$ do nó n_i para o nó n_j e $\boldsymbol{v} \in defg(n_i)$ }; *defg*(n_i) representa o conjunto de todas as variáveis com definição global no nó n_i)

(iv) **t**-*uso(j,k)*={Variável $\boldsymbol{t}$ | $\boldsymbol{t}$ possui um uso nos arcos (n_j,n_k) sempre que um nó $n_j \in N_s$ tiver um dos comandos INSERT, UPDATE, DELETE ou SELECT (arco de saída). O exemplo da Figura 2 possui um ***t*-*uso*** nos arcos (2,3) e (2,4) com relação a ***t***.

Dizemos que o caminho $(n_\alpha,..., n_\beta,...,n_j, n_k)$ é um caminho livre de *definição persistente* do par de nós $<n_\alpha, n_\beta>$ até o nó n_k, se não existir outro par de nós $<n_q, n_m>$ onde ocorre uma redefinição persistente de *t* em $<n_q, n_m>$ do nó n_β até o nó n_k onde o par $< n_\alpha, n_\beta>$ antecede o par $<n_q, n_m>$. Na ausência de um comando **COMMIT**, após um comando executável da **SQL** até o último nó de um grafo, adotou-se que o **COMMIT** estará no último nó do grafo. Essa é uma estratégia de implementação.

As seguintes definições complementam os conceitos básicos para a definição dos critérios de teste de programas de ABDR:

(v) Associação definição-t-uso é uma tripla $[<n_\alpha,n_\beta>, (n_j,n_k), \boldsymbol{t}]$ onde $\boldsymbol{t} \in defT<n_\alpha,n_\beta>$, $n_j \in dsu(\boldsymbol{t}, \beta)$, o arco (n_j, n_k) é o arco de saída do nó n_j.

(vi) dtu-caminho é um caminho livre de *definição persistente* $(n_\alpha, ..., n_\beta, ..., n_j, n_k)$ c.r.a. $\boldsymbol{t}$ dos nós $<n_\alpha , n_\beta>$ até o nó n_k ou até o arco (n_j, n_k) onde ocorre um uso de $\boldsymbol{t}$ e o caminho $(n_\alpha, ..., n_\beta, ..., n_j, n_k)$, onde $n_\beta < n_j$ e $n_\beta < n_k$ é um caminho livre de laço e no par de nós $<n_\alpha, n_\beta>$ ocorre uma *definição persistente* de $\boldsymbol{t}$. Um *dtu-caminho* é um *du-caminho* com relação à *variável tabela* no enfoque de variável persistente.

3. Teste de Integração

Esta seção introduz conceitos básicos de teste de integração, a partir de relações de fluxo de dados em variáveis tabela.

3.1 Teste de integração baseado na dependência de dados

A principal característica de programas de aplicação, que os distingue dos programas convencionais, é a persistência dos dados da base de dados. As unidades de programas e os módulos de programa, se autorizados, podem ter acesso a esses dados a qualquer momento. A persistência é uma propriedade que não existe exclusivamente em Banco de Dados Relacional (BDR), podendo existir para outros programas com características semelhantes.

Em ABDR, a mesma tabela da base de dados pode estar disponível para diferentes módulos de programa e, conseqüentemente, para diferentes *UPs* de um mesmo módulo. Neste caso, duas unidades de programa podem possuir uma dependência de dados entre elas, mesmo não existindo chamadas conectando-as.

Duas unidades de programa UP_A e UP_B, não necessariamente distintas, possuem uma dependência de dados de UP_A para UP_B com relação à *variável tabela* $\boldsymbol{t}$, caso a unidade UP_A possua um ponto no programa que define a *variável tabela* $\boldsymbol{t}$ (isto é, altera o conteúdo da tabela cujo estado passa de e_i para e_{i+1}) e a unidade de programa UP_B possua um ponto no programa que usa a *variável tabela* $\boldsymbol{t}$ (com o estado de $\boldsymbol{t}$ igual a e_{i+1}). Uma *variável tabela* $\boldsymbol{t}$ com uma *definição persistente* em uma unidade UP_A (passando a variável $\boldsymbol{t}$ para um estado e_A) estará em um estado consistente (representado por e_A) se e somente se não existir *redefinição persistente* de $\boldsymbol{t}$ em pelo menos um subprograma UP_A no nó de saída. Existe uma dependência de dados c.r.a. *variável*

tabela ***t*** se ***t*** tiver uma definição persistente em uma unidade UP_A e existir um caminho livre de definição c.r.a. ***t*** incluindo o sub-caminho de UP_A a partir da definição persistente até o nó de saída e um sub-caminho do nó de entrada de uma unidade UP_B até o arco (n_j, n_k) que contém um ***t-uso***.

São considerados dois tipos de dependência de dados c.r.a ***t***:

i) ***Dependência interna ou intra-modular***: ocorre quando existir uma dependência de dados entre unidades de programa de um mesmo Módulo, com relação a uma ou mais tabelas, mesmo quando não existir um ponto de chamada entre elas (Figura 3).

ii) ***Dependência externa ou inter-modular***: ocorre quando existir uma dependência de dados entre duas ou mais *UPs* de Módulos diferentes c.r.a ***t*** . Isto é, ***t*** deve estar disponível para os dois Módulos (Figura 4).

O Grafo de Dependência de dados de um programa de ABDR é denotado por $\boldsymbol{GD_{PA}(A_{Modi} \times B_{Modj})_T} = ([G(A_i) \times G(B_j)], E_{int}, T)$, onde $G(\boldsymbol{A_i}) \times G(\boldsymbol{B_j})$ representa a união dos conjuntos de arcos e nós do grafo de programa da unidade **A** do módulo de programa ***i*** e do grafo de programa da unidade **B** do módulo de programa ***j***; *E* é o conjunto de arcos (setas pontilhadas) que mostram o fluxo de dados das unidades de programa para a tabela e vice-versa, e *T* é o conjunto de tabelas envolvidas na dependência de dados. $\boldsymbol{GD_{PA}(A_{Modi} \times B_{Modj})_t}$ é um multi-grafo direcionado que representa o fluxo de dados baseado na dependência de dados extraída da associação def-uso c.r.a. ***t***. Um exemplo desse grafo é apresentado na Figura 4.

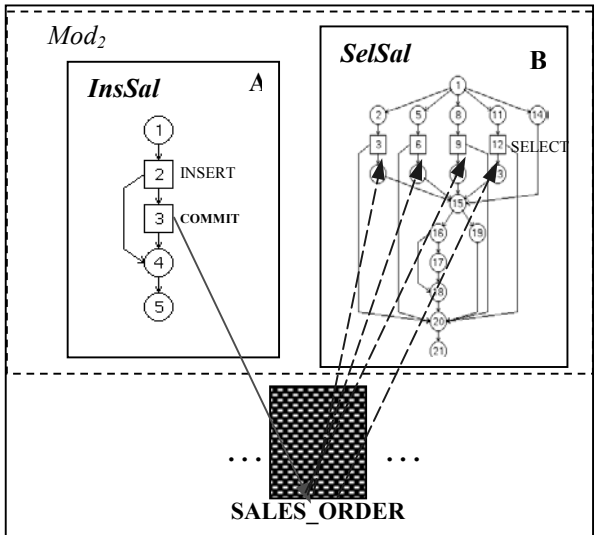

Figura 3: Grafo de dependência Interna de dados c.r.a *SALES_ORDER*.
$\boldsymbol{GD_{PA}(InsSal_{Mod3} \times SelSal_{Mod3})_{SALES\_ORDER}}$

Nas Figuras 3 e 4, uma *seta* (continua) direcionada da *UP* para a tabela caracteriza a ocorrência de uma *definição* persistente da variável tabela ***t*** pela *UP*; uma seta (pontilhada) direcionada da tabela ***t*** para a *UP* caracteriza a ocorrência de um *uso* da tabela ***t*** pela *UP*. O exemplo mostrado na *Figura 3* é um grafo de dependência *Interna*, contudo se as *UPs A* e *B* pertencerem a Módulos de Programas distintos tratar-se-á de grafo de dependência *Externa* (Figura 4). No exemplo da Figura 3, a *UP InsSal* define a variável tabela *SALES_ORDER* (nó 3) e a *UP SelSal* usa a variável tabela *SALES_ORDER* em vários pontos do programa (nós 3, 6 e 9). Isso é feito para todas as UPs que possuem uma definição persistente, combinadas com todas as *UPs* que usam a mesma variável tabela. Já no exemplo da Figura 4 a *UP UpdEmp* do Módulo de Programa Mod_3 define a variável

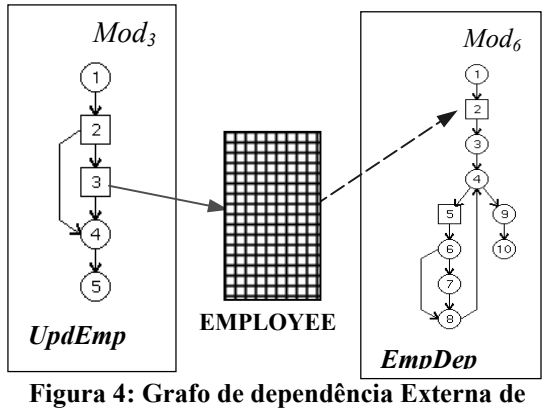

Figura 4: Grafo de dependência Externa de dados c.r.a *EMPLOYEE* entre as *Ups UpdEmp* (Mod_3) e *EmpDep* (Mod_6).
$\mathbf{GD_{PA}(UpdEmp_{Mod3} \times EmpDep_{Mod6})_{EMPLOYEE}}$

tabela EMPLOYEE no nó 3 e a *UP EmpDep* do Módulo de Programa Mod_6 usa essa mesma variável no nó 2.

4. Tipos de Fluxos de Dados

A análise de fluxo de dados para programas de ABDR, que é composta por dois tipos de Fluxos de Dados - intra-modular e inter-modular - explora as associações persistentes geradas pelas variáveis tabela. Esta seção apresenta as definições básicas e a terminologia para a definição de critérios de teste estrutural de integração que enfocam as variáveis tabela utilizadas em programas de ABDR.

4.1. Fluxo Intra-modular

O fluxo de dados *intra-modular* é utilizado para o teste de integração de cada Módulo de Programa. Inicialmente é feito o teste de cada *UP* que compõe o Módulo de Programa (teste *intra-unidade* ou teste de unidade) [SPO00a]. Depois que todas as Unidades foram devidamente testadas, realiza-se o teste de integração para as *UPs* do Módulo de Programa (teste *inter-unidades* ou teste de integração *intra-modular*).

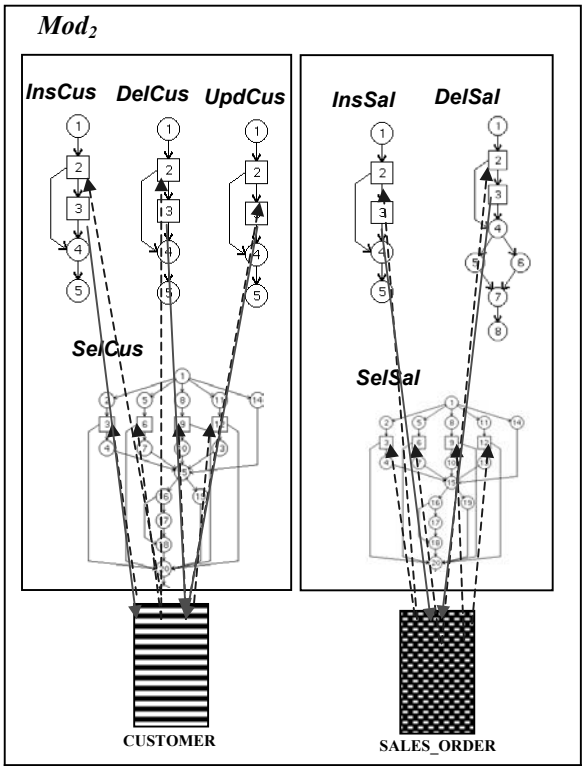

Figura 5: Exemplo do Fluxo de Dados *Intra-Modular* completo.

A partir do grafo de programa *G(UP)* de cada *UP* do Módulo é construído o grafo de dependência de dados que representa os fluxos de dados entre as *UPs* que compõem o Módulo. O grafo de dependência baseia-se na hierarquia de chamada entre as *UPs* ou na dependência de dados entre as *UPs* determinada pelas *variáveis tabela* presentes no Módulo.

A Figura 5 apresenta um exemplo completo de dependência de dados entre as unidades de um mesmo módulo (Mod_2). O Módulo Mod_2 possui comandos de **SQL** com definições e usos *persistentes* de duas das oito *variáveis tabela* mostradas na Figura 6. Toda seta direcionada da *variável tabela* ***t*** para o nó de um grafo representa um uso de ***t*** naquele nó e toda seta apontada do nó para a *variável tabela* ***t*** representa uma *definição persistente* de ***t*** naquele nó. Toda unidade que tem um *uso* de uma variável ***t*** possui uma dependência de dados com as unidades que têm uma *definição persistente* da mesma variável ***t***.

4.2. Fluxo Inter-modular

O fluxo de dados *inter-modular* é determinado a partir das dependências de dados ocasionadas pelas tabelas da base de dados comuns aos Módulos da ABDR. A unidade UP_i no *Módulo* Mod_k pode estar *definindo* uma *variável tabela* ***t*** enquanto a unidade UP_j contida no *Módulo* Mod_W, $K \neq W$, pode estar *usando* a mesma *variável tabela* ***t***,

determinando, assim, um fluxo de dados entre os módulos Mod_k e Mod_w. Assim, é possível definir associações *definição-uso* c.r.a ***t*** *(*denominada de *associação definição-t-uso-inter*) entre os módulos de programas criando um grafo de dependência externa de dados entre a UP_i do Módulo Mod_k, que possui uma *definição persistente* da variável ***t***, e a UP_j do Módulo Mod_w, que possui um *uso* da variável ***t***. A representação gráfica do modelo de fluxo de dados *inter-modular* é composta por nós retangulares contínuos que representam os *Módulos de Programa* e nós retangulares pontilhados, representando as *variáveis tabela* que compõem a base de dados da aplicação. As setas indicam os fluxos de dado entre os nós. Tendo em vista que um Módulo de Programa é visto como um conjunto de Unidades de Programa $Mod_a = \{UP_1, .., UP_m\}$, temos:

a) Seta do *Módulo de Programa* (Mod_i) para tabela ***t***: indica que existe pelo menos uma $UP_\alpha \subset Mod_i$ tal que UP_α tem uma definição *persistente* da tabela ***t***;
b) Seta da tabela ***t*** para o *Módulo de Programa* Mod_i: indica que existe pelo menos uma $UP_\alpha \subset Mod_i$ tal que UP_α tem um uso *persistente* da tabela ***t***.

No fluxo *inter-modular*, a execução de cada módulo de programa é feita isoladamente, não havendo, portanto, rotinas de controle entre eles. De acordo com o exemplo apresentado na Figura 6, o módulo de programa Mod_2 define e usa as tabelas CUSTOMER e SALES_ORDER. É possível determinar as associações *definição-uso* inter-modular para cada *Módulo de Programa* (*Mod*); essas informações são extraídas das *UPs* pertencentes a cada *Módulo de Programa* que possuírem um par $<n_\alpha, n_\beta>$, da *definição persistente* de ***t***, com as *UPs* que possuírem um comando da **SQL** que usa a tabela ***t***.

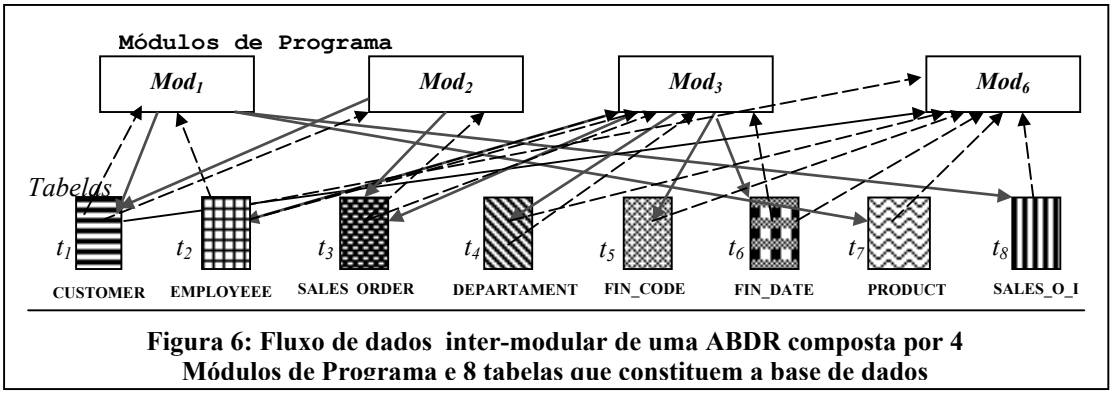

Figura 6: Fluxo de dados inter-modular de uma ABDR composta por 4 Módulos de Programa e 8 tabelas que constituem a base de dados

Tabela 1: Mapa de definição e uso das variáveis t_i mostradas nos Módulos de Programas da Figura 6

Módulos de Programa	Tabelas *definidas*								Tabelas *usadas*							
	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
Mod_1	X	X					X	X	X							
Mod_2	X		X						X		X					
Mod_3		X	X	X	X	X				X	X	X	X	X		
Mod_6									X	X	X	X	X	X	X	X

Para melhor entendimento da Figura 6, é apresentado na Tabela 1 um mapa que relaciona para cada Módulo de Programa, as ações de ocorrência (definição e uso) para cada variável tabela (numeradas da esquerda para a direita).

4.3. Os critérios de integração intra-modulares

Os critérios de teste intra-modulares são aplicados no teste de cada UP do Módulo (teste de unidade) e no teste de integração das UPs do Módulo. Para o teste de unidade

podemos utilizar qualquer critério de teste estrutural. Os critérios para o teste de integração são divididos em dois subgrupos: critérios de testes baseados no grafo de chamadas e critérios de teste baseados na dependência de dados das tabelas.

Os critérios de teste propostos para associar as *variáveis persistentes* devem ser satisfeitos com a mesma tupla, forçando o testador a gerar casos de testes específicos para exercitá-la. Deste modo, o teste deve ser executado de maneira controlada, não sendo suficiente usar qualquer tupla de ***t***. Por outro lado, a exigência do uso da mesma tupla para satisfazer uma *associação definição-t-uso*, pode aumentar o número de elementos requeridos não factíveis (isto é, não executáveis com a mesma tupla).

- ✓ **Critérios de teste de integração baseados na dependência de dados**

Os critérios de integração *intra-modular* visam a exercitar as *associações definição-t-uso* determinadas pelos comandos de manipulação da base de dados. Os critérios de integração baseados na dependência de dados determinada pelas tabelas da base de dados visam a exercitar as associações referentes às *variáveis de programa* (***P***), *variáveis host* (***H***) ou *variáveis tabela* (***T***), de acordo com o enfoque estabelecido pelos critérios.

A *variável tabela* definida por uma unidade de programa mantém-se viva ao longo do tempo, determinando a existência de uma associação *def-t-uso* entre duas *UPs* executadas em momentos distintos, sem a necessidade de um comando de chamada de uma para outra. Dizemos que uma associação *def-t-uso* existe, se uma variável ***t*** definida em UP_A tem um *t-uso* em UP_B.

As associações *definição-t-uso* de integração são exercitadas com seqüências de execuções para pares de unidades (UP^d, unidade que define ***t***; UP^u, unidade que usa ***t***). Neste caso, é importante definir uma estratégia que exercite essas associações, utilizando os mesmos valores, tanto para a execução de UP^d como para a de UP^u.

- ✓ **Critérios baseados no fluxo de dados intra-modular**

Estes critérios baseiam-se nas dependências de dados existentes entre os procedimentos de um mesmo *Módulo de Programa* com relação a uma *variável tabela* da base de dados e requerem a mesma *tupla* para satisfazer a associação *definição-t-uso*. Cada unidade UP_A que contém uma *definição persistente* da variável ***t*** é associada a outra unidade UP_B que contém um *uso* de ***t (t-uso)***; os pares (UP_A, UP_B) são requeridos pelo critério com um ciclo de execução (denomina-se de *ciclo1*). A noção de ciclo 1 denota que somente as UPs que definem a associação serão executadas para cobrir a associação; em alguns casos, como dependência por chave estrangeira, pode ser que outras UPs necessitem ser executadas, requerendo mais ciclos de execução para cobrir a associação.

Considere que $G(UP_A)$ e $G(UP_B)$ sejam os grafos das unidades A e B, respectivamente, envolvidas na integração e Π seja o conjunto dos caminhos completos em $G(UP_A)$ e $G(UP_B)$ e Γ o conjunto de tuplas. $(\pi_a, \pi_b) \in \Pi$ implica que existe a concatenação π_a, π_b, onde $\pi_a \in G(UP_A)$ e $\pi_b \in G(UP_B)$.

Todas os t-usos-ciclo1-intra: Π e Γ satisfazem o critério *todos os t-usos-ciclo1-intra* para o par de unidades UP_A e UP_B pertencentes ao mesmo módulo se, para todos os pares de caminhos $(\pi_a, \pi_b) \in \Pi$: π_a incluir o par de nós $<n_\alpha, n_\beta> \in G(UP_A)$ tal que $defT< n_\alpha, n_\beta> \neq \phi$ c.r.a. ***t***; existir um caminho livre de definição c.r.a. ***t*** de $<n_\alpha, n_\beta>$ até n_{out} de $G(UP_A)$; π_b incluir o arco $(n_j, n_k) \in G(UP_B)$ onde $j \in fdsu(t, n_\beta)$; existir um ***t-uso***

em (n_j, n_k); e existir um *caminho livre de definição persistente* c.r.a. $\boldsymbol{t}$ de n_{in} até o arco (n_j, n_k); a associação é satisfeita se for exercitada através da mesma tupla $\tau \in \Gamma$.

Uma *associação-t-uso-ciclo1-intra* para o módulo z é representada pela quíntupla:

$$[^{z}UP_A, <n_\alpha, n_\beta>, {}^{z}UP_B, (n_j, n_k), \boldsymbol{t}]$$

Todos os dtu-caminhos-intra: Π e Γ satisfazem o critério *todos os dtu-caminhos-intra* para o par de unidades UP_A e UP_B pertencentes ao mesmo módulo se, para todos os pares de caminhos $(\pi_a, \pi_b) \in \Pi$: π_a incluir um *dtu-caminho* do nó n_α até o nó n_{out} de $G(UP_A)$ passando pelo par de nós $<n_\alpha, n_\beta> \subseteq G(UP_A)$ onde $defT<n_\alpha, n_\beta> \neq \phi$ c.r.a. $\boldsymbol{t}$, através da tupla τ, π_b incluir um *dtu-caminho* c.r.a $\boldsymbol{t}$ do nó n_{in} de $G(UP_B)$ *até o arco* $(n_j, n_k) \in G(UP_B)$ sendo $n_j \in fdsu(\boldsymbol{t}, n_\beta)$*;* existir um ***t-uso*** em (n_j, n_k), através da mesma tupla τ; e existir um *caminho livre de definição persistente* que vai do par $<n_\alpha, n_\beta>$ em UP_A até o arco (n_j, n_k) em UP_B c.r.a $\boldsymbol{t}$ na composição dos caminhos $\pi_a . \pi_b$.

O elemento requerido pelo critério *todos os dtu-caminhos-intra* é representado pela n-upla abaixo, sendo z o número do módulo que contém as unidades A e B:

$$[^{z}UP_A \ n_\alpha \ a_1 \ ... \ a_n \ n_\beta \ ... \ n_{out} - {}^{z}UP_B \ n_{in} \ ... \ n_j, n_k],$$

onde $^{z}UP_A$ é a unidade onde ocorre a definição de $\boldsymbol{t}$ e $n_\alpha \ a_1 \ ... \ a_n \ n_\beta \ ... \ n_{out}$ é o *dtu-caminho* de $<n_\alpha, n_\beta>$ até o nó de saída, podendo ou não passar por um sub-caminho $(a_1 \ ... \ a_n)$ livre de laço; $^{z}UP_B$ é a unidade onde ocorre o uso de $\boldsymbol{t}$ e $n_{in} \ ... \ n_j \ n_k$ é o *dtu-caminho* que vai do nó inicial até o arco (n_j, n_k) onde existe um *t-uso* c.r.a. $\boldsymbol{t}$.

Existem dependências que não podem ser exercitadas com a execução de apenas duas unidades (uma de *definição* e outra de *uso*), mas necessitam da execução de outra unidade para satisfazer a associação. Isso ocorre com a existência da integridade referencial entre duas tabelas, ou seja, para definir a variável $\boldsymbol{t}$ é necessário estar definida a variável $\boldsymbol{t'}$ e, para isso, pode ser necessário executar a unidade que define a variável $\boldsymbol{t'}$ para depois executar a unidade que define $\boldsymbol{t}$ e finalmente executarmos a *UP* que usa $\boldsymbol{t}$.

Todos os t-usos-ciclo2-intra: Π e Γ satisfazem o critério (*todos os t-usos-ciclo2-intra*) para as unidades UP_A, UP_B e UP_C pertencentes ao mesmo módulo se, para todos os caminhos $(\pi_a, \pi_b, \pi_c) \in \Pi$, onde π_a é o caminho que contém um par de nós $< n_{\alpha'}, n_{\beta'}>$ que *define* a variável $\boldsymbol{t'}$ pela *tupla* τ'; o caminho π_b contém um par de nós $<n_\alpha, n_\beta>$ que *define* $\boldsymbol{t}$ pela tupla τ podendo ou não ter uma dependência de $\boldsymbol{t'}$, *e* o caminho π_c possuir um nó que contém um *uso* de $\boldsymbol{t}$ e ou $\boldsymbol{t'}$ pela mesma *tupla* τ *e/ou* τ', respectivamente, e o caminho π_a passar pelo par de nós $<n_{\alpha'}, n_{\beta'}> \in G(UP_A)$ com $defT<n_{\alpha'}, n_{\beta'}> \neq \phi$ c.r.a. $\boldsymbol{t'}$: existir um *caminho livre de definição* c.r.a. $\boldsymbol{t'}$ de n_β até n_{out} de $G(UP_A)$; o caminho π_a passar pelo par de nós $<n_\alpha, n_\beta> \in G(UP_B)$ onde $defT<n_\alpha, n_\beta> \neq \phi$ c.r.a. $\boldsymbol{t}$; existir um *caminho livre de definição* c.r.a. $\boldsymbol{t}$ de n_β até n_{out} de $G(UP_B)$*;* o caminho π_b passar pelo arco $(n_j, n_k) \in G(UP_C)$/ existe um ***t-uso*** em (n_j, n_k) e $n_j \in fdsu(\boldsymbol{t}, n_\beta)$ (sendo $\boldsymbol{t}$ e/ou $\boldsymbol{t'}$) e existir um *caminho livre de definição* c.r.a. $\boldsymbol{t}$ e/ou $\boldsymbol{t'}$ de n_{in} até o arco (n_j, n_k). A associação $[<n_{\alpha'}, n_{\beta'}>, UP_{\boldsymbol{A}}, <n_\alpha, n_\beta>, UP_{\boldsymbol{B}}, (n_j, n_k), UP_C, \{\boldsymbol{t', t}\}]$ é satisfeita se e somente se a mesma *tupla* τ *(* τ'*)* usada para satisfazer a *definição* de $\boldsymbol{t}$ ($\boldsymbol{t'}$) é também usada para satisfazer o *uso*.

Neste caso, os elementos requeridos são representados por uma sétupla do tipo:

$$[^{z}UP_A, <n_\alpha, n_\beta>, {}^{z}UP_B, <n_{\alpha'}, n_{\beta'}>, {}^{z}UP_C, (n_j, n_k), \{\boldsymbol{t, t'}\}]$$

Dois enfoques podem ser considerados: utilizar as mesmas tuplas para satisfazer os pares *definição-uso* na integração ou utilizar tuplas diferentes para satisfazer os elementos requeridos não cobertos com a mesma tupla. O primeiro enfoque é mais conservador e mostrou, na execução de um exemplo de aplicação, ser melhor para a detecção de defeitos do que o segundo; contudo, este enfoque possui maior custo associado, pois a quantidade de elementos requeridos tende a ser maior.

4.4. Critérios de integração inter-modular

Os critérios de integração inter-modular são semelhantes aos critérios de integração intra-modular com a diferença de que as unidades associadas devem pertencer a Módulos distintos. Esses critérios visam a exercitar associações de variáveis tabela que são definidas em unidades de um Módulo α e são usadas em unidades de outro Módulo β.

Considere duas unidades UP_A, pertencente a um módulo Mod_x, e UP_B, pertencente a outro módulo Mod_y. Considere também os mesmos conjuntos $\boldsymbol{\Pi}$ e $\boldsymbol{\Gamma}$ definidos anteriormente. Temos:

Todos os t-usos-ciclo1-inter: Este critério é similar ao critério *todos os t-usos-ciclo1-intra*, com a única diferença que as unidades UP_A e UP_B pertencem a Módulos distintos. Neste caso o elemento requerido é representado pela quíntupla:

$$[^{z}UP_A\ ,\ <n_\alpha,\ n_\beta>,^{w}UP_B\ ,\ (n_j,\ n_k)\ ,\ \mathbf{t}]$$

onde *z* representa o módulo Mod_z e *w* representa Mod_w.

Todos os dtu-caminhos-inter: Este critério é similar ao critério *todos os dtu-caminhos-intra*, com a única diferença que a unidade onde existe uma definição persistente (UP_A) pertence a um Módulo e a unidade onde existe um ***t-uso*** pertence a outro Módulo. Neste caso o elemento requerido é representado pela quíntupla:

$$[^{z}UP_A\ \ n_\alpha\ \ a_1\ \ldots\ a_n\ \ n_\beta \ldots\ \ n_{out}\ -\ ^{w}UP_B\ \ n_{in}\ \ldots\ n_j\ \ n_k]$$

onde *z* representa o módulo Mod_z e *w* representa Mod_w.

Todos os t-usos-ciclo2-inter: Este critério é similar ao critério *todos os t-usos-ciclo2-intra*, com a diferença que as unidades *associadas* devem pertencer a Módulos distintos. Neste caso, os elementos requeridos são representados por uma sétupla do tipo:

$$[^{z}UP_A\ ,\ <n_\alpha,\ n_\beta>,\ ^{y}UP_B,\ <n_{\alpha'},\ n_{\beta'}>,\ \ ^{w}UP_C\ ,\ (n_j,\ n_k)\ ,\ \{\mathbf{t},\ \mathbf{t'}\}]$$

4.5. Aplicação dos Critérios Propostos

Os critérios propostos foram utilizados no teste de integração em aplicações reais, para revelar defeitos existentes e semeados em laboratório. Para tal, a Ferramenta POKE-TOOL [CHA91], originalmente voltada à automação de teste de fluxo de dados de programas convencionais, foi adaptada para apoiar a aplicação dos critérios de integração para ABDRs.

Para exemplificar, considere o teste de integração intra-modular baseado na dependência de dados, para a variável tabela EMPLOYEE no Módulo de Programa denominado de Mod3. Os elementos requeridos dos critérios todos os t-usos-ciclo1-intra e todos os dtu-caminhos-intra são mostrados na Figura 7.

Para ilustrar, <**3001,<2,3>,3016,(2,4),{EMPLOYEE}**> é a associação entre as unidades **3001** e **3016**; no par **<2,3>** ocorre a definição persistente da variável **EMPLOYEE**, na unidade **3001**; e no arco **(2,4)** ocorre o uso da variável **EMPLOYEE**

na unidade **3016**. O mesmo ocorre com os elementos requeridos para o critério todos os *dtu-caminhos-intra* representados pelas seqüências **3011 2 3 4 5** e **3016 1 11 12 18**; a primeira é o *dtu-caminho* **(2, 3, 4, 5)** na unidade **3011** referente à definição-persistente de **EMPLOYEE**; a segunda é o *dtu-caminho* **(1, 11, 12, 18)** na unidade **3016** referente ao uso da variável persistente.

```
ASSOCIACOES REQUERIDAS PELOS CRITERIOS
TODAS DEF-T-USOS-INTRA EMPLOYEE
Associacoes requeridas pelo Grafo(3001)
- insemp
1) <3001,<2,3>, 3001,(2,3),{EMPLOYEE}>
2) <3001,<2,3>, 3001,(2,4),{EMPLOYEE}>
. . .
14)<3001,<2,3>,3016,(12,18),{EMPLOYEE}>
. . .
Associacoes requeridas pelo Grafo(3011)
29) <3011,<2,3>,3001,(2,3),{EMPLOYEE}>
30) <3011,<2,3>,3001,(2,4),{EMPLOYEE}>
. . .
42)3011,<2,3>,3016,(12,18),{EMPLOYEE}>
```

```
ASSOCIACOES REQUERIDAS PELOS CRITERIOS
TODOS DTU-PATHS-INTRA - EMPLOYEE
Caminhos Requeridos pelo
Grafo(3001) - insemp
01) 3001 2 3 4 5 - 3001 1 2 3
02) 3001 2 3 4 5 - 3001 1 2 4
. . .
14) 3001 2 3 4 5 - 3016 1 11 12 18
. . .
Caminhos Requeridos pelo Grafo(3011)-updemp
29) 3011 2 3 4 5 - 3001 1 2 3
30) 3011 2 3 4 5 - 3001 1 2 4
. . .
42) 3011 2 3 4 5 - 3016 1 11 12 18
```

Figura 7: Elementos requeridos pelos critérios todos os *t-usos-ciclo1-intra* e *todos os dtu-caminhos-intra*.

Na Figura 8 enumeram-se os caminhos percorridos pela aplicação dos casos de teste e o valor da chave primária ou conjunto de chaves utilizadas em cada execução. Por exemplo, as execuções dos casos de testes #2 e #3 exercitaram o elemento requerido *15*, mencionado na Figura 7, do critério *todos-t-usos-ciclo1-intra*, onde em ambos foi manipulada a mesma tupla, identificada por 1900.

A Tabela 2 apresenta o total de elementos requeridos (elem.req.) pelo critério Todos os *t-usos-ciclo1-intra*, para as variáveis tabela associadas ao Módulo Mod3. Apresenta também o total de elementos requeridos não exercitáveis (ne) e o número de casos de teste para exercitar os elementos requeridos exercitáveis. A Tabela 3 mostra o resultado para tabelas associadas nos testes de integração (*intra-modular* e *inter-modular*) dos Módulos para os critérios de *ciclo1* e *ciclo2*.

Pathint.tes

```
#1
3001   1   2   3   4   5

#2
3006   1   2   3   4   5

#3
3001   1   2   3   4   5

#4
3011   1   2   3   4   5

#5
3001   1   2   4   5

#6
3001   1   2   3   4   5

#7
3001   1   2   4   5
. . .
```

. #número do caso de teste
. número da unidade e seus respectivos caminhos executados

Keyint.tes

```
#1
3001
EMPLOYEE: 1900

#2
3006
EMPLOYEE: 1900

#3
3001
EMPLOYEE: 1900

#4
3011
EMPLOYEE: 1900

#5
3001
EMPLOYEE: 1900
. . .
```

número do caso de teste
número das unidades (3xxx)
variável e tuplas utilizadas (ch primária)

Figura 8: *I*nformação sobre a execução do teste de integração *intra-modular*.

Não existe elemento requerido para o teste de integração *intra-modular* do Módulo Mod_6, tendo em vista que este programa emite apenas relatórios e usa diversas tabelas em seu código de SQL. Na integração *inter-modular*, existem 48 elementos re-queridos em cada ciclo para o critério *todos os t-usos* (sendo 42 com Mod_3 e 6 com o Mod_2). Os defeitos ocasionados pelas queries durante as consultas só foram observados com a integração entre os módulos que associaram com o Mod_6.

O número total (234) de casos de teste para o teste de integração (critério *todos-t-usos-ciclo1-intra*) do Módulo Mod_3 (Tabela 2), necessário para exercitar os elementos requeridos, deve-se à exigência de uso da mesma tupla no critério de teste de integração.

Unidades de programas que somente emitem relatórios (possuem apenas comandos SELECT) não tiveram elementos requeridos durante o teste de unidade no escopo de variáveis persistentes. Contudo, na combinação de unidades, foram identificados elementos requeridos pelos critérios de integração.

Foi observado que os critérios de integração intra-modular e de integração inter-modular não podem ser comparados quanto a sua força relativa, ou seja, quanto à dificuldade em satisfazê-los. Em adição, tais critérios enfocam aspectos distintos de programas de uma ABDR, tornando-os, então, incomparáveis teoricamente. A aplicação dos critérios Potenciais Usos não destaca os defeitos referentes ao uso dos comandos da SQL, sendo estes observados somente com os critérios baseados na dependência de dados *intra-modular* e *inter-modular*.

Tabela 2: Testes de integração critério todos os *t-usos-ciclo1-intra* para o Mod_3

Tabelas	*Total de elem.req. (ne)*	*N^{os} de Casos de testes*
Departamento	36 (9)	60
Employee	42 (10)	65
Fin	48(24)	60
Sales_Order	36(10)	49

Tabela 3: Elementos requeridos pelo critério *todos os t-usos-inter* (Mod_2, Mod_3 e Mod_6)

Mod_{def}	Mod_{uso}	*Elementos Requeridos Todos os t-usos*		*Casos de testes*	
		Ciclo1	*ciclo2*	*ciclo1*	*Ciclo2*
Mod_2	Mod_3	**9**	**-**	**18**	**-**
Mod_2	Mod_6	**6**	**6**	**12**	**18**
Mod_3	Mod_2	**9**	**-**	**18**	**-**
Mod_3	Mod_6	**42**	**42**	**40**	**60**

5. Conclusões

Foram exploradas as relações de fluxo de dados persistentes, em aplicações de banco de dados relacional (ABDR), para abstrair requisitos de teste estrutural de integração. O objetivo foi suprir a carência de critérios de teste estrutural de integração para ABDR, envolvendo o uso da SQL (structured query language) para a manipulação de dados persistentes.

Foi proposta uma abordagem aos testes estruturais de integração *intra-modular* e *inter-modular* em programas com **SQL** embutida. Tais critérios são baseados na estrutura de chamada entre unidades e na dependência exclusiva de dados, ambos focados nas relações de fluxo de dados persistentes.

A criação de novas estratégias de teste estrutural, visando a exercitar aplicações de Banco de Dados, complementa as técnicas de teste estrutural já existentes na literatura e os novos critérios contribuem com novas abordagens de teste que podem ser aplicadas a variáveis persistentes de um modo geral. Os critérios propostos, ao exigirem o uso da mesma tupla para satisfazer as *associações definição-t-uso*, distinguem-se da maioria dos demais critérios de teste; isto obriga o testador a escolher os dados de teste com mais cuidado e resulta, ao mesmo tempo, em um teste mais efetivo no que se refere à integridade das tabelas.

A partir de exemplos de aplicação foi possível observar que os critérios são complementares, pois possuem requisitos próprios por abordar estruturas e conceitos distintos, tais como diferentes tipos de fluxo de dados. Em adição, dependendo da natureza da integração entre unidades e módulos existentes na aplicação, o emprego de uma das abordagens de integração pode ser suficiente durante a atividade de teste; por exemplo, o Mod_6 (ver Figura 6), o qual é um programa que usa somente as variáveis tabela, pode ter todos os seus comandos SQL exercitados com o teste de integração *inter-modular* mostrado na Figura 4.

A realização de estudos experimentais controlados, visando a avaliar o custo, a aplicabilidade e a propensão em revelar defeitos dos critérios propostos, será o próximo passo direcionado à melhoria da qualidade de ABDRs.

Agradecimento: Agradecemos ao *CNPq* e *Capes* pelo apoio financeiro.

6. Referências

[CHA91] Chaim, M. L. "Poke-tool – Uma Ferramenta para Suporte ao Teste Estrutural de Programas Baseado em Análise de Fluxo de Dados", Dissertação de Mestrado, DCA/FEE/UNICAMP, Campinas – SP – Brasil, Abril 1991.

[CHA00] Chays, D., Dan, S., Frankl, P.G., Vokolos, F.I., Weyuker, E.J. "A Framework for Testing Database Applications". Proceedings of the ISSTA´00-, Agosto, 2000, pp. 147-157.

[CHA03] Chays, D., Deng, Y., "Demonstration of AGENDA Tool Set for Testing Relational Database Applications", Proc. of The Intl. Symposium on Software Engineering, Porland, Oregon, 2003.

[CLA89] Clarke, L. A., Podgurski, A., Richardson, D. J., Zeil, S.J., "A Formal Evaluation of Data Flow Path Selection Criteria", IEEE TSE, 15(11), Novembro, 1989, pp. 1318-1332.

[LEI05] Leitão, P. S. J., Vilela, P. R. S., Jino, Mario, "Mapping Faults to Failures in SQL Manipulation Commands", ACS/IEEE Intl. Conference on Computer Systems and Applications, Cairo, Egypt, 3-6 January, 2005.

[MAL88] Maldonado, J. C.; Chaim, M. L., Jino, M. "Seleção de Casos de Testes Baseada nos Critérios Potenciais Usos", II SBES, Canela, RS, Brasil, Outubro, 1988, pp.24-35.

[MAL91] Maldonado, J. C., "Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software", Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, SP, Brasil, Julho, 1991.

[RAP82] Rapps, S., Weyuker, E.J., "Data Flow Analysis Techniques for Test Data Selection", in Intl. Conf. on Soft. Eng., pp. 272-278, Tokio, Japan, Setembro., 1982.

[RAP85] Rapps, S., Weyuker, E.J., "Selection of Software Test Data Using Data Flow Information", IEEE TSE, SE-11(4), Abril, 1985.

[SPO00a] Spoto, E. S., Jino, M., Maldonado, J. C. "Teste Estrutural de Software: Uma abordagem para Aplicação de Banco de Dados Relacional", SBES'00, XIV SBES, João Pessoa – PB, pp. 243-258, Outubro, 2000.

[SPO00b] Spoto, E. S., "Teste Estrutural de Programas de Aplicação de Banco de Dados Relacional", Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, SP, Brasil, Dezembro, 2000.

[WU03] Wu, X., Wang,Y., Zheng, Y. "Privacy Preserving Database Application Testing", ACM – WPES´03, October, 2003 – Washington – USA.