5
Avaliação experimental

5.1.
Cenário

No capítulo 1, seção 1.2, foi apresentado o cenário deste estudo de caso que motivou esta dissertação. Fazendo uma recapitulação, tudo se iniciou numa parceria entre o LES e o TecGraf, outro laboratório da PUC-Rio, que desenvolvia e dava manutenção a um software com cinco anos de existência. Esse sistema possuía toda a sua lógica de negócio escrita na linguagem *PL/SQL* do SGBD Oracle. Havia 150.000 linhas de código espalhadas em 340 *procedures*. A interface de acesso era feita em Java, sendo esta uma camada muito fina, apenas para exibir o retorno das *procedures* chamadas em cada funcionalidade acessível pelo usuário. Neste contexto, o LES teria basicamente duas funções com esta parceria: desenvolver testes unitários e documentar o sistema.

Com relação ao desenvolvimento de testes, não houve problemas técnicos. A grande dificuldade era o entendimento de como as *procedures* se comunicavam e como funcionava o fluxo de chamadas.

A única documentação existente era o documento Word exibido na Figura 1, que consistia de um código copiado e caixas de textos com explicações sobre a árvore de chamadas daquela *procedure*. Esta documentação apenas ajudava o entendimento do funcionamento daquela *procedure* em especial, porém tinha uma série de limitações:

- Este documento não existia para todas as *Procedures*. Apenas algumas eram assim documentadas.
- A geração do documento era feito manualmente. Logo, não existia nenhuma garantia de estar atualizado.
- Não ajudava na análise de impacto, pois não era possível ter uma visão de quem utilizava aquela *procedure*. Apenas quem ela utilizava.
- Não tinha uma visão por tabela.

PUC-Rio - Certificação Digital Nº 0621307/CA

- A árvore de chamadas era escrita por um ser humano. Estava sujeita a falhas e erros de entendimento.

Inspirado nos problemas acima surgiu a idéia de desenvolver um aplicativo que conseguisse resolver as falhas na documentação existentes. Esta concepção deu origem a esta dissertação.

5.2. Processo

Como explicado anteriormente, a ferramenta necessita da entrada de arquivos de metadados para funcionar plenamente

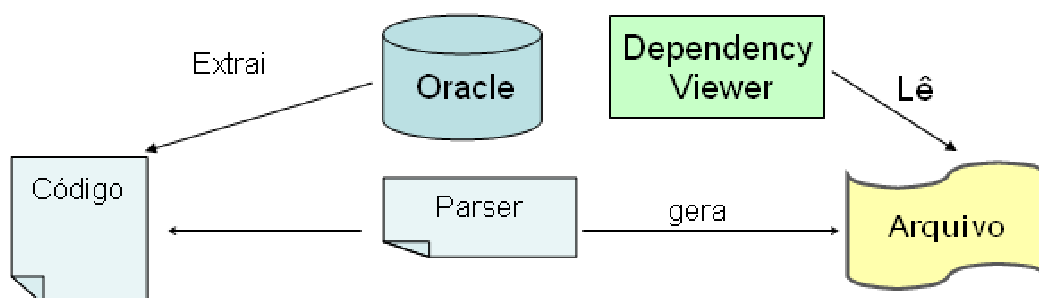

Figura 31: Processo de geração da documentação automatizada

Toda a lógica de negócio estava armazenada em um sistema gerenciador de banco de dados (SGBD) Oracle na forma de *Procedure*. Para fazer uso do *Dependency Viewer*, era necessário criar um *parser* para ler este SGBD e produzir um arquivo de entrada na linguagem descritiva entendida pela ferramenta. No início cogitou-se a implementação de um compilador simples, que teria como objetivo extrair a árvore de chamadas e as informações necessárias diretas do Oracle. Verificou-se, com o início da implementação, que isso seria uma tarefa árdua. Cada versão do Oracle tinha mudanças significativas na gramática da linguagem *PL/SQL*. Isto significava que a cada mudança de versão, seria necessária uma adaptação para que a ferramenta funcionasse corretamente. Devido a isso, e a maior facilidade de implementação, foi utilizada a abordagem de extração da informação através de padrões léxicos.

Para tanto, era necessário gerar um arquivo de texto com todo o código do software, extraído do Oracle. Foi utilizado, então, o aplicativo *SqlDeveloper* [Oracle 2009] para esta finalidade. Ele possui uma opção de exportar uma cópia do banco para um arquivo texto.

PUC-Rio - Certificação Digital Nº 0621307/CA

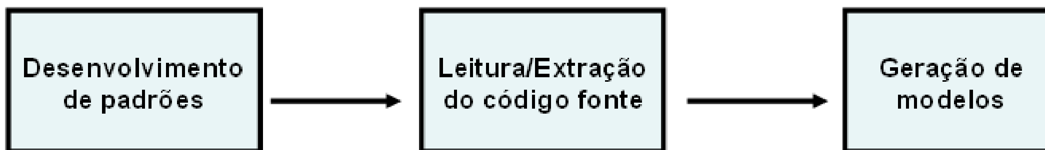

Figura 32: Processo de recuperação do modelo

Como visto na figura acima, o processo de recuperação do modelo da estrutura de software proposto possui as seguintes etapas:

- **Encontrar padrões para a extração de informações (I):** Consiste em identificar o que precisa ser extraído do código fonte e encontrar padrões léxicos que possibilite isso.
- **Gerar uma ferramenta para extrair do código fonte os padrões criados (II)**: Este passo consiste em produzir um *parser* que consiga interpretar os padrões criados e gerar arquivos de entrada para a ferramenta *Dependency Viewer*.
- **Gerar os modelos (III):** Esta fase consiste basicamente em alimentar a ferramenta, que ela se encarrega de gerar os modelos. Foi facilitada pela existência da mesma.

5.2.1. Padrões

Um padrão deve buscar uma característica no código e deve ter um objetivo específico no processo de recuperação da estrutura. Neste trabalho, a geração de padrões buscou encontrar os relacionamentos estruturais do sistema, baseando-se em um exame da sintaxe da linguagem PL/SQL, num estilo de programação, e nos comandos SQL padrão. Assim, foram criados cinco padrões:

- Relacionamentos entre *procedures*.
- Relacionamento entre *procedures* e tabelas
- Relacionamento entre tabelas
- Extração de comentários
- Extração de parâmetros de entrada e saída da *procedure*

Depois da definição dos padrões, desenvolveu-se uma série de programas para extraí-los do código fonte. Esses programas foram desenvolvidos na linguagem AWK [Aho et al. 1988], que tem como principal objetivo a transformação de arquivos. O AWK conta com recursos para encontrar padrões (*pattern matching*) e para manipulação de expressões regulares. Tais recursos

PUC-Rio - Certificação Digital Nº 0621307/CA

foram determinantes para o sucesso da mineração de relacionamentos descritos nos padrões. Desta forma, com base no código (passado para um arquivo .txt) e nos padrões estabelecidos criaram-se programas em AWK que fizeram a engenharia reversa, recuperando os dados necessários. Os programas AWK geram arquivos com os dados necessários para se recuperar os modelos do sistema. Esses arquivos são organizados – um *parser* – para que a ferramenta *Dependencies Viewer* (Capítulo 3) possa ler e apresentar toda a informação na forma gráfica.

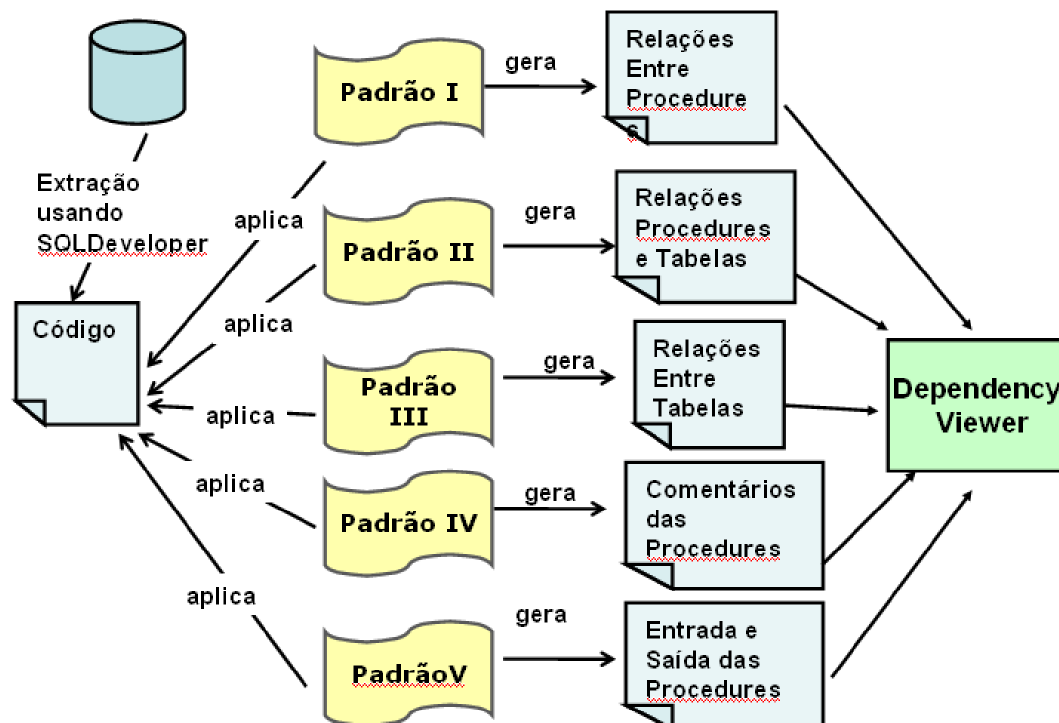

Figura 33: Ligação dos padrões com a ferramenta *Dependency Viewer*

Na Figura 33 é mostrado um esquema de como funcionam os padrões e as suas relações com a ferramenta. Cada padrão é traduzido numa ferramenta. Essas rodam em cima de um arquivo textual contendo o código fonte da aplicação. Este arquivo foi extraído do Oracle através do aplicativo *SqlDeveloper*. Cada ferramenta produz um arquivo de saída que serve de entrada para o *Dependency Viewer*.

Segue abaixo um detalhamento de cada padrão.

PUC-Rio - Certificação Digital Nº 0621307/CA

5.2.1.1.
Relacionamento entre *procedures* (I)

Para o reconhecimento desses relacionamentos foi necessário descobrir todos os pacotes e *procedures* existentes no código PL/SQL. Para tanto, fez-se: (i) a eliminação dos comentários, tanto expressos com "/* ... */" quanto com "--"; (ii) a busca de dados através dos seguintes padrões: "PACKAGE/package", "*procedure*" e "function", e; (iii) formatação dos dados recuperados para que a lista de saída tivesse os pacotes e as suas respectivas *procedures*. Após a identificação dos pacotes e *procedures*, deve-se procurar os relacionamentos entre *procedures*, que são caracterizados pelas seguintes expressões:

- "_pkg.<nome_proc2>": representa um relacionamento entre a *procedure* que foi encontrada e a *procedure* referenciada no padrão "_pkg.<nome_proc2>"; Isso só é possível, pois todos os pacotes do sistema terminam com o sufixo "_pkg". É uma regra imposta pelos desenvolvedores do software, e não pela linguagem PL/SQL.
- "pacote.procedure": cada palavra do código que atendesse a esse padrão foi selecionada e comparada na listagem produzida anteriormente em (iii). Se houver a ocorrência na listagem então adiciona-se um relacionamento entre a *procedure* onde o padrão foi recuperado e a *procedure* da listagem.

```
procedure GET_STOCK_DATA ()
begin

 common_products_pkg.filter_spm_products(null)

 common_products_pkg.update_products_info(present_date)

common_const_pkg.TF_TRUE); -- Para atualizar o

    set_of_balance_type := stock_pkg.get_single_types();
...
```

Figura 34: Exemplo do padrão I

5.2.1.2.
Relacionamento entre *procedures* e tabelas (II)

O padrão definido para extrair estes relacionamentos verifica os comandos que contém as expressões *update*, *delete*, *from*, *join* e *insert*.

PUC-Rio - Certificação Digital Nº 0621307/CA

```
procedure compute_actual_data_density() is
  begin

    update temp_itens_realizacao
    set densidade = massa / volume
    where volume <> 0
          and massa <> 0

end compute_actual_data_density;
```

Figura 35: Exemplo do padrão II

5.2.1.3. Relacionamento entre tabelas (III)

O padrão para extrair estes relacionamentos verifica os comandos que iniciam com *alter table* ou *create table* e que contém as expressões *foreign key* e *references*.

```
ALTER TABLE "ANOTACAO_CENARIO"
ADD CONSTRAINT "ANCE_CENA_FK"
FOREIGN KEY ("CENA_NR_NUMERO_CENARIO")
 REFERENCES "CENARIO" ("CENA_NR_NUMERO_CENARIO") ENABLE;
```

Figura 36: Exemplo do padrão III

5.2.1.4. Comentários das *procedures (IV)*

O padrão para extrair os comentários busca os caracteres de início e fim de comentário dentro do PL/SQL ("/*" e "*/").

PUC-Rio - Certificação Digital Nº 0621307/CA

```
/*
    Retorna dados de realizacao de determinados tipos de balanco.

    param set_of_balance_type: conjunto de tipos de balanco
    param grouping: indica se e' para retornar valor diario ou
mensal.
*/
  procedure OPEN_ACTUAL_DATA_CURSOR
  (  set_of_balance_type in set_of_balance_type_type,
    grouping             in int,
    actual_data_cursor   out cursor_type   )
```

Figura 37: Exemplo do padrão IV

5.2.1.5.
Parâmetros de entradas e as saídas das *procedures (V)*

O padrão para mostrar as entradas e saídas busca por: (i) sentenças "PACKAGE/package"; (ii) comandos iniciados com as palavras-chaves "procedure" ou "function", e; (iii) comandos contendo as expressões "in" ou "out".

```
procedure OPEN_ACTUAL_DATA_CURSOR   (
        set_of_balance_type in set_of_balance_type_type,
        grouping            in int,
        actual_data_cursor  out cursor_type   )
```

Figura 38: Exemplo do padrão V

5.2.1.6.
Prova de conceito

Para mostrar a aplicabilidade da abordagem, a ferramenta foi utilizada para extrair o modelo da estrutura de um sistema legado de médio porte, escrito em PL/SQL. Nesta seção, será mostrada a extração de relacionamentos entre procedures e entre procedures e tabelas e a geração do grafo.

O Código 9 mostra um trecho do código da *procedure update_scenarium* que possui um relacionamento com uma tabela cenário.

```
procedure UPDATE_SCENARIUM
  (...) is ...
  begin
    select ce.cena_nr_numero_cenario, ce.cena_dt_criacao
    into scena_cd_id, creation_date
    from cenario ce
    where ce.cena_nr_numero_cenario = to_number(scenarium_id.identifier);
    ...
  end;
```

Código 9: Trecho 1 da *procedure update_scenarium*

PUC-Rio - Certificação Digital Nº 0621307/CA

O trecho marcado acima mostra o código que indica um relacionamento entre a *procedure* e uma tabela. A ferramenta encontra o código a partir do padrão II e gera o trecho mostrado na Figura 39 no arquivo da estrutura.

```
27  SCENARIUM_PKG.actually_comp_closing_w_oprog-> <from> tipo_balanco
28  SCENARIUM_PKG.actually_comp_closing_w_oprog-> <from> tipo_balanco
29  SCENARIUM_PKG.UPDATE_SCENARIUM-> <from> cenario
30  SCENARIUM_PKG.UPDATE_SCENARIUM-> <update> cenario
31  SCENARIUM_PKG.APPROVE_SCENARIUM-> <from> cenario
```

Figura 39: Trecho gerado pela aplicação do padrão II

Já o Código 10 mostra um trecho do código da *procedure update_scenarium* que possui um relacionamento com outras *procedures*.

```
procedure UPDATE_SCENARIUM
  (...) is ...
  begin
    ...
    if open_stock.count > 0 then update_open_stock(open_stock);
    end if;
    if delivery.count > 0 then update_delivery(delivery);
    end if;
    ...
    if fcc_processing.count > 0 then ...
    ...
  end;
```

Código 10: Trecho II da procedure update_scenarium

O trecho marcado no código acima mostra o código que indica um relacionamento entre a *procedure* e duas outras *procedures* (*update_open_stock* e *update_delivery*). Esses trechos são encontrados pela ferramenta a partir do padrão I. O arquivo gerado (parcial) após o processamento do código está mostrado na Figura 40 (se a *procedure* é chamada várias vezes ela se repete, no entanto depois isto é filtrado).

```
66  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_production
67  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_imp_exp
68  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_imp_exp
69  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_target_stock
70  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_transfer
71  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_load_unload
72  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_load_unload
73  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_op_program
74  scenarium_pkg.UPDATE_SCENARIUM->scenarium_pkg.update_annotation
75  scenarium_pkg.APPROVE_SCENARIUM->scenarium_pkg.get_source_scena_creation_date
76  scenarium_pkg.APPROVE_SCENARIUM->scenarium_pkg.create_scenarium
```

Figura 40: Trecho gerado pela aplicação do padrão I

Depois de aplicar todos os padrões, a ferramenta gera o seguinte grafo de relacionamentos para a procedure *update_scenarium* (Figura abaixo).

PUC-Rio - Certificação Digital Nº 0621307/CA

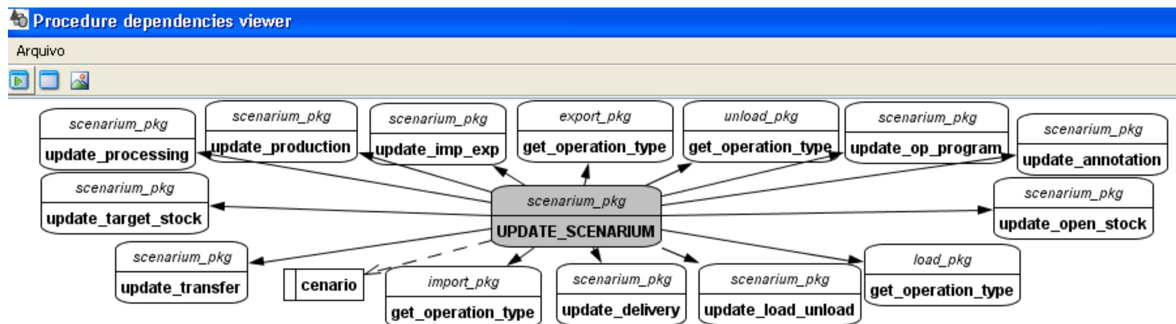

Figura 41: Modelo da estrutura recuperado a partir da *procedure update_scenarium*

5.3. Avaliação

Após todo o processo e definições mostrados neste capítulo, a ferramenta *Dependency Viewer* ficou pronta para ser usada no projeto do LES. Uma mudança na parceria levou o LES a assumir novas responsabilidades no projeto. O laboratório ficou responsável não apenas pelo desenvolvimento de testes e documentação, mas também pela manutenção das *procedures*. Isso ocorreu num momento em que a pessoa responsável por esta atividade, e a que mais conhecia o domínio e o código da aplicação, se desligou do projeto e da empresa parceira. Neste contexto, o benefício que a ferramenta *Dependency Viewer* traria ganhou mais importância e notoriedade, como pode ser visto pelo depoimento de Alexandre Almeida, desenvolvedor do LES.

> “A análise de impacto de mudanças em um sistema é um fator delicado e que deve ser bem mensurado. Nesse quesito a "ferramenta" foi de suma importância para nos ajudar na analise do impacto das mudanças com segurança e rapidez.” (Alexandre Almeida)

A ferramenta está sendo usada pela equipe de testes e manutenção do sistema faz seis meses. Abaixo segue um resumo de alguns benefícios que ela comprovadamente trouxe, e exemplos de cenário de uso.

5.3.1. Análise de impacto de uma *procedure*

A ferramenta *Dependencies Viewer* auxilia o desenvolvedor a fazer uma análise de impacto de um pedido de mudança. A Figura 42 mostra um cenário de uso da ferramenta. Imagine que um desenvolvedor, atendendo a um pedido de mudança, saiba que será necessário modificar uma determinada *procedure*

PUC-Rio - Certificação Digital Nº 0621307/CA

(neste exemplo, chamada de *get_scenarium*). O desenvolvedor usa a ferramenta para saber se existem outras *procedures* que chamam a *procedure* em questão, encontrando a *procedure open_scenarium* (Figura 42.1). Neste ponto, o desenvolvedor pode recuperar as procedures diretamente relacionadas à *procedure* open_scenarium (Figura 42.2).

Caso seja necessário, o desenvolvedor pode verificar se existem outras chamadas indiretas para a *procedure*. É possível, assim, visualizar toda a hierarquia de chamadas – opção "Todos os níveis – para open_scenarium (Figura 42.3). Essa verificação apresenta diversos relacionamentos. O retângulo cinza representa a *procedure* utilizada para a busca (na parte superior se tem o nome do pacote e na parte inferior o nome da *procedure*). Além da análise dos relacionamentos entre procedures, poder-se-ia analisar as tabelas associadas a cada *procedure* impactada. Estes diagramas mostram um "mapa" da implementação. Em alguns casos, têm-se muitos caminhos e, quando se tem pouco conhecimento sobre o software, o esforço para a análise é bem maior. Informações associadas a cada procedure/tabela (semelhante a um Java.doc) ajudam no entendimento das funções de cada um destes elementos.

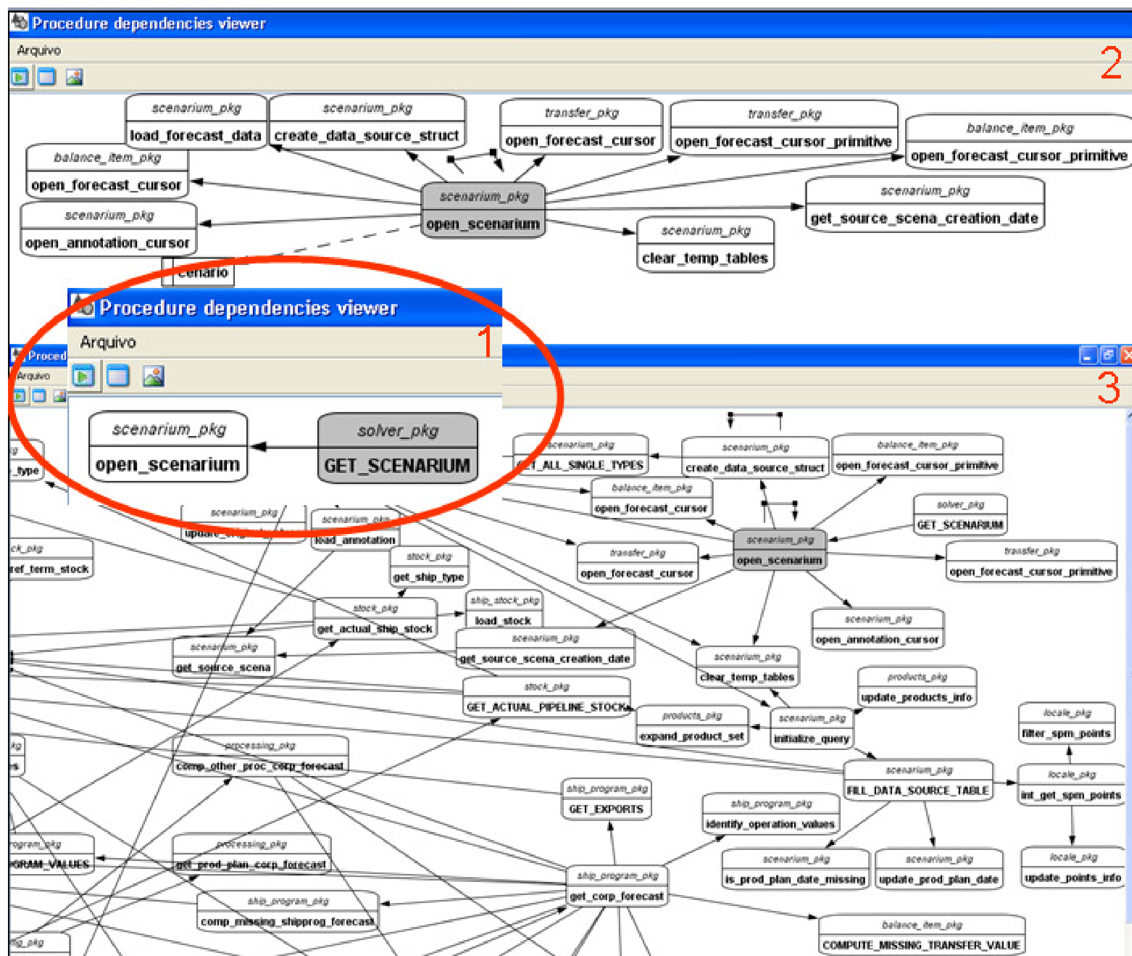

Figura 42: Navegando para realizar a análise de impacto para mudança numa *procedure*

PUC-Rio - Certificação Digital Nº 0621307/CA

5.3.2. Análise de impacto por tabela

Na seção acima, foi mostrada a análise de impacto a partir da visão de uma *procedure*. É comum também usar esta análise a partir da visão de uma tabela, e a ferramenta auxilia nisso. A Figura 43 mostra um cenário de uso imaginando ser preciso uma mudança na função de permissão. Para isso, seria necessário alterar a tabela **Permissão**. Usando a ferramenta, descobrem-se quais são as *procedures* que removem e inserem dados nesta tabela (Figura 43.1). nesse caso a *procedure remove_folder* remove dados, e a *procedure save_other_user_permissions* insere dados. Apesar de o nome das *procedures* remeter às suas funções, estas informações foram retiradas através do estereótipo da ligação das *procedures* com as tabelas. O desenvolvedor pode, também, ter uma visão de todas as *procedures*, ou outras tabelas, diretamente relacionadas com a tabela **Permissão** (Figura 43.2).

Similar à visão a partir de uma *procedure*, na Figura 43.3 é possível ter uma visão global do mapa de implementação permitindo uma análise de impacto completa.

Figura 43: Navegando para realizar a análise de impacto para mudança numa tabela

PUC-Rio - Certificação Digital Nº 0621307/CA

PUC-Rio - Certificação Digital Nº 0621307/CA

5.3.3. Análise de tabelas temporárias

Como dito anteriormente, toda a regra de negócio estava espalhada em 340 *procedures*. Um meio comum de comunicação entre elas é através de passagem de parâmetros por tabelas temporárias. Observe a Figura abaixo, retirada da ferramenta *Dependency Viewer*. A tabela **temp_cenarios** é um exemplo deste caso. Ela é criada e populada pela *procedure initialize_stock_query*, que é chamada pela outra *procedure insert_scena_closing,* que também lê a tabela temporária ***temp_cenarios.*** Como pode ser observado em destaque na Figura 44, este é um típico caso no qual uma *procedure* cria e popula uma tabela temporária para outra *procedure* ler as informações mais adiante.

Imagine que o desenvolvedor esteja alterando a *procedure initialize_stock_query*, que cria a tabela temporária **temp_cenarios**. Ele sente a necessidade de mudar um campo desta tabela, ou até mesmo não mais criá-la, por achar que não é usada. Através da ferramenta, ele pode ver que esta tabela é lida por outras *procedures* do sistema, portanto possui o seu papel ativo e não pode ser alterada sem uma análise detalhada prévia.

Em um outro cenário de uso, imaginando o desenvolvedor dando manutenção na *procedure insert_scena_closing.* Ele nota, em algum momento do código, que ela lê informações da tabela temporária **temp_cenarios**. Percebe, além disso, que estas informações estão erradas. O desenvolvedor teria que varrer todo o código atrás da *procedure* que popula esta tabela. Com a ferramenta estas informações podem ser vistas rapidamente no diagrama gerado.

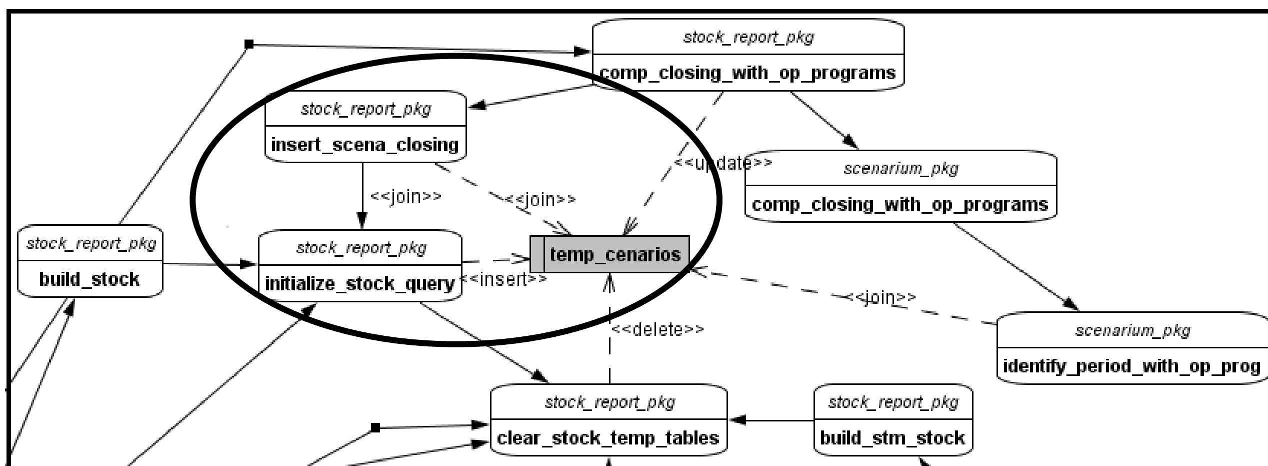

Figura 44: Pedaço do diagrama gerado para a tabela **temp_cenarios**

Esse é um cenário de uso muito comum para a ferramenta, dado que são muitos estes casos ao longo do sistema. Ela se mostrou útil em ajudar nesta análise.

5.3.4. Entendimento do Sistema

Imaginando um cenário em que o desenvolvedor tenha que dar manutenção na *procedure create_view*, porém não faz idéia da razão da existência desta. Ele pode ir à ferramenta *Dependency Viewer* e gerar o diagrama da Figura 45.

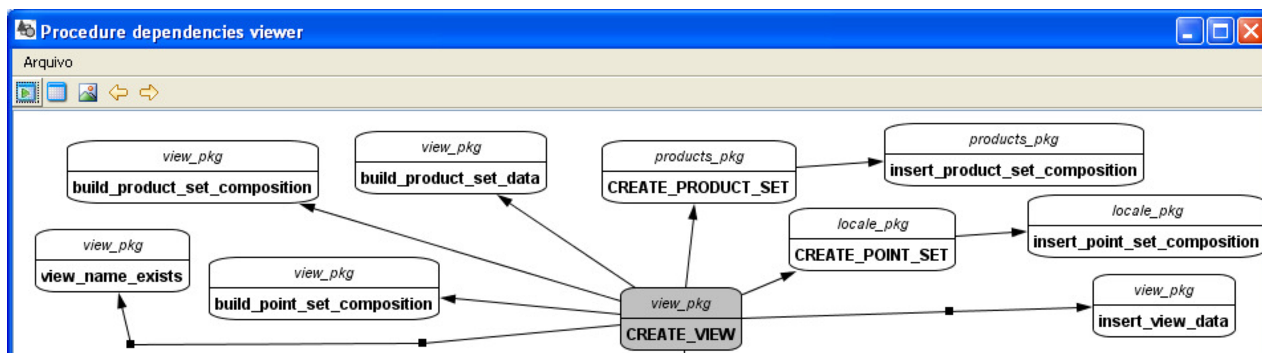

Figura 45: Diagrama com a *procedure create_view* como referência inicial.

Olhando a figura acima, ele reconhece outras *procedures* e as suas funções. Percebe rapidamente que se trata de um código que está relacionado com estoque de produtos, e passa a ter mais segurança em fazer a alteração. Esse é outro cenário comum de uso confirmado, por depoimentos, como sendo corriqueiro e parcialmente resolvido pela ferramenta.

PUC-Rio - Certificação Digital Nº 0621307/CA

5.3.5.
A metaferramenta

O estudo experimental focou resolver o problema apresentado na motivação desta dissertação. Para isso, foi preciso apenas realizar uma única customização da ferramenta *DependencyViewer*. Toda a avaliação foi baseada nesta instância única. Vale ressaltar, porém, que a ferramenta pode ser adaptada para diversos outros domínios. Ela foi concebida para servir como meta-ambiente de visualização para outros projetos voltados para a área de engenharia reversa.

Para poder provar a sua capacidade metamórfica e a sua versatilidade, foi criada outra instância de uso, com exatamente a mesma versão da ferramenta usada no estudo de caso. A única diferença entre as duas são os metadados de configuração e as informações de modelo.

Foi concebido um novo domínio de negócio baseado na avaliação experimental existente. A ferramenta agora deve exibir apenas o modelo relacional do banco de dados estudado, e toda a interface deve orientar o usuário para deixa claro isso.

Fazendo um passo a passo de como customizar a ferramenta, a primeira tarefa é refazer o arquivo *dependenciesViewer.config*.

```
configuration {
	application[
		title="Modelo Relacional";
		files={"table.txt"}
	];
	"tabela"[
		id=Nome,
		label="Tabela",
		shape="com.coutosistemas.shapes.enums.BoxShapeEnum"
	];
	edge[
		source="table",
		target="table",
		targetTerminator="none"
	]
	menu[
		title="Escolha a tabela",
		node="procedure",
		columns={"name","Nome da Tabela"},
	]
}
```

Código 11: Novo arquivo de configuração

No Código 11, é apresentado como ficou o novo arquivo de configuração. Como num modelo relacional a tabela é exibida em um retângulo simples, foi usado a classe *BoxShapeEnum* fornecida pelo *framework* gráfico (sessão 4.2). É importante ressaltar que o usuário conhecendo bem a API do *framework* poderia facilmente customizar o seu próprio desenhador.

PUC-Rio - Certificação Digital Nº 0621307/CA

O próximo passo é a criação do arquivo de modelo. Foi gerado, então, um arquivo contendo todos os relacionamentos entre tabelas. Ele exibe como estereótipos de origem e destino a chave estrangeira e o atributo identificador, respectivamente. No código abaixo é mostrado parcialmente como fica este arquivo.

```
diagram{
	"table"-"table"[
		["UNMA_CD_ID"]"PARAMETRO_ESTOQUE_AGREG"->"UNIDADE_MEDIDA_ABT" ["UNMA_CD_ID"]
		["PROD_CD_ID"] "PARAMETRO_ESTOQUE_PROD"->"PRODUTO" ["PROD_CD_ID"]
		["UNMA_CD_ID"] "PARAMETRO_ESTOQUE_PROD"->"UNIDADE_MEDIDA_ABT" ["UNMA_CD_ID"]
		["ACAO_CD_ID"] "PERMISSAO"->"ACAO" ["ACAO_CD_ID"]
		                        (...)
		["USER_ID"] "PERMISSAO"->"USUARIO_BANDEIRA" ["USER_ID"]
		["DIRE_CD_ID"] "PERMISSAO_DIRETORIO"->"DIRETORIO" ["DIRE_CD_ID"]
		["USER_ID"] "PERMISSAO_DIRETORIO"->"PERMISSAO" ["USER_ID"]
		["USER_ID"] "PERMISSAO_VISAO"->"PERMISSAO" ["USER_ID"]
	]
}
```

Código 12: Arquivo de modelo

O Código 12 foi gerado a partir de uma adaptação das ferramentas criadas para o estudo experimental.

Com esses dois arquivos prontos, a ferramenta está funcional dentro do novo modelo de negócio.

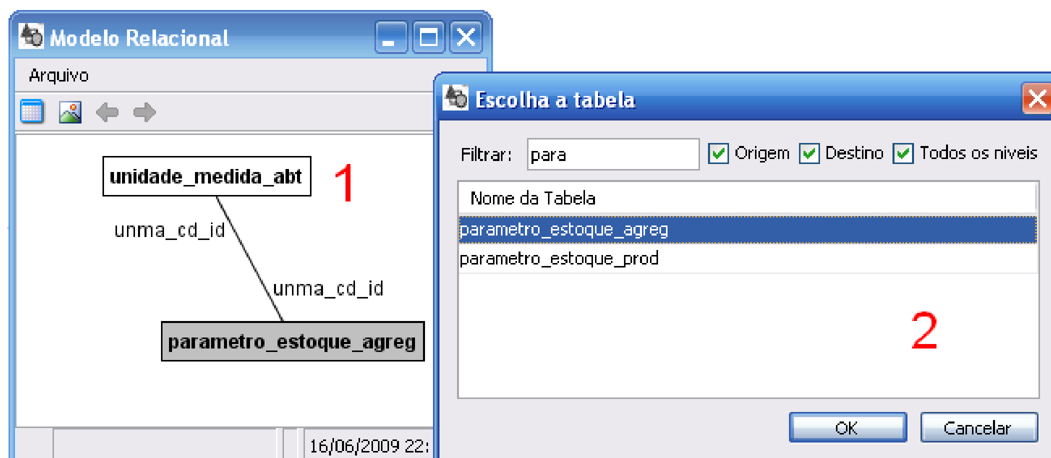

Figura 46: Cara da nova instância da ferramenta

Na Figura 46.1 é apresentado como ficou o diagrama gerado pela nova configuração da ferramenta. Na Figura 46.2 é mostrada a tela de busca por uma tabela.

Imaginando um cenário de uso do aplicativo, o usuário pode, como mostra a Figura 47.1, escolher uma tabela e gerar um diagrama completo. A maioria das ferramentas que fazem este trabalho de engenharia reversa não fornece uma opção de seleção de um subgrafo do diagrama maior gerado, portanto seria realmente uma documentação difícil de ser entendida. Na Figura 47.2, é

PUC-Rio - Certificação Digital Nº 0621307/CA

desmarcada a opção **Todos os Níveis**. Isso gera um diagrama menor, com o centro sendo a tabela alvo do estudo. Com a opção de navegação, o usuário pode facilmente navegar por grafos pequenos, que não fornecem uma visão geral, porém provêem um melhor entendimento do funcionamento do sistema, devido à simplificação do modelo. Esta opção não exclui ainda a possibilidade de geração do modelo completo e imprimi-lo, como fazem outras ferramentas do gênero.

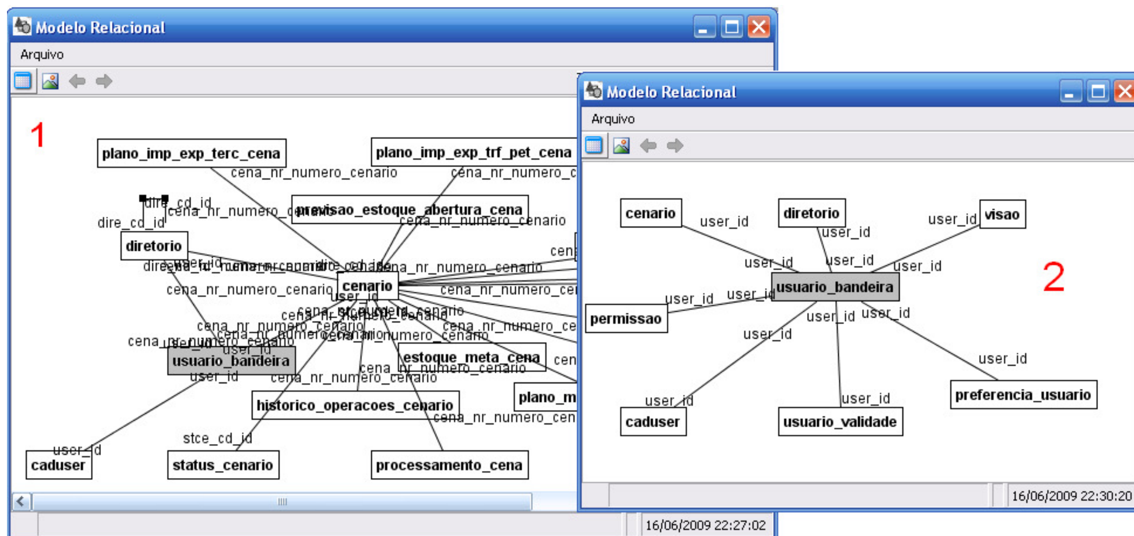

Figura 47: Análise do modelo

Esse caso prova que é possível gerar aplicações funcionais para diferentes domínios mudando apenas os arquivos de metadados.

PUC-Rio - Certificação Digital Nº 0621307/CA