

Universidade Federal de Pernambuco

graduação em Ciência da Computação

Centro de Informática

**PL/SQL Helper: Uma ferramenta de auxílio à manutenção de sistemas em PL/SQL**

**Aluno:** Marcelo Victor Carneiro Pereira (mvcp@cin.ufpe.br)

**Orientadora:** Valéria Cesário Times (vct@cin.ufpe.br).

Julho de 2009

**Resumo**

Este trabalho aborda a especificação e desenvolvimento de uma ferramenta de software capaz de prover funcionalidades para fins de manutenção de sistemas de bancos de dados desenvolvidos em linguagem PL/SQL. Com a pesquisa, foi verificada a falta de ferramentas com esse objetivo e a ausência de materiais que especifiquem modelos ou arquiteturas de software para esta finalidade.

**Palavras-chave:** sistemas de gerenciamento de bancos de dados, análise, manutenção, PL/SQL, padrões de projeto, diagrama de dependência, refatoramento, testes, documentação.

**Índice**

**1 Introdução....................................................................................................................................5**

* 1. Motivação........................................................................................................................5
  2. Objetivos.........................................................................................................................6
  3. Estrutura do Documento...................................................................................................7

1. **Fundamentação Teórica...........................................................................................................8**
   1. Introdução......................................................................................................................8
   2. Conceitos Básicos............................................................................................................9
      1. Rotina...........................................................................................................9
      2. Pacotes..........................................................................................................10
      3. Pós-codificação..............................................................................................10
      4. Métricas........................................................................................................10
   3. Técnicas de Engenharia de Software...............................................................................11
      1. Padrões de Projeto.........................................................................................11
      2. Refatoramento...............................................................................................12
      3. Teste de Software...........................................................................................14
      4. Documentação de Software............................................................................16
   4. Ambientes PL/SQL........................................................................................................17
   5. Trabalhos Correlatos......................................................................................................18
   6. Conclusão......................................................................................................................20
2. **Especificação da Ferramenta PL/SQL Helper.........................................................................21**
   1. Introdução....................................................................................................................21
   2. A arquitetura do PL/SQL Helper......................................................................................21
   3. As funcionalidades do PL/SQL Helper............................................................................23
      1. Documentação...............................................................................................24
      2. Análise........................................................................................................25
         1. Gráfico de Dependências.....................................................................26
         2. Métricas..............................................................................................27
         3. Diagrama Relacional...........................................................................27
      3. Refatoramento...............................................................................................27
         1. Fachada.............................................................................................28
         2. Renomeação........................................................................................28
      4. Testes.............................................................................................................29
   4. Comparativo.................................................................................................................30
   5. Conclusão....................................................................................................................30
3. **Protótipo do PL/SQL Helper...................................................................................................32**
   1. Introdução....................................................................................................................32
   2. Funcionalidades.............................................................................................................33
      1. Inicialização..................................................................................................33
      2. Conexão.......................................................................................................34
      3. Documentação..............................................................................................35
      4. Análise..........................................................................................................36
      5. Refatoramento..............................................................................................38
   3. Conclusão.....................................................................................................................39
4. **Estudo de caso........................................................................................................................40**
   1. Projeto SIG@...............................................................................................................40
   2. Implantação.................................................................................................................41
   3. Resultados....................................................................................................................41
   4. Conclusão.....................................................................................................................47
5. **Conclusões.............................................................................................................................48**
   1. Contribuições................................................................................................................48
   2. Trabalhos Futuros.........................................................................................................49

6.3 Referências Bibliográficas.............................................................................................50



**1 Introdução**

Este capítulo possui três seções. Inicialmente, na seção 1.1, será dada a motivação para realização deste trabalho, listando-se as principais dificuldades encontradas no domínio de aplicação de bancos de dados que serviram de motivação para o estudo realizado. Em seguida, na seção 1.2, serão expostos os objetivos do desenvolvimento do sistema proposto neste documento e por fim, a estrutura deste documento e a definição sobre como ele está organizado são dadas na seção 1.3.

**1.1 Motivação**

O desenvolvimento de um software, atualmente, é um processo complexo, com um grande número de etapas e diversas metodologias a serem seguidas. As principais etapas podem ter nomes diferentes de autor para autor, mas em geral, são definidas como: análise dos requisitos, especificação do sistema, codificação do sistema, teste de software, documentação e manutenção. Cada uma delas possui diversos padrões e recebem uma atenção especial do grupo desenvolvedor para que sejam obtidos melhores resultados no final.

As estratégias de desenvolvimento de software e suas abordagens podem ser agrupadas em metodologias de desenvolvimento de software. Scrum [45], Programação Extrema [46] e o Processo Racional Unificado (RUP) [47] podem ser citados como exemplos dessas metodologias. Para o ambiente de bancos de dados, tais metodologias e padrões para as etapas de desenvolvimento de aplicações de BD, não existem na literatura, acarretando em uma execução não-padronizada da implementação do software.

Alguns dos principais padrões de desenvolvimento de software como o *Observer Template* e *Dependency Injection* já foram implementados para ambientes de bancos de dados [14, 15, 16] para alcançar melhorias de produtividade, legibilidade e manutenção de código, mas em geral, são pouco difundidos e não foram definidos especificamente para a arquitetura de sistemas de bancos de dados. Tais padrões, apesar de importantes, não servem para outras etapas do ciclo de desenvolvimento, tais como testes, análise do sistema e documentação, por exemplo.

Um desenvolvedor PL/SQL possui poucas ferramentas especiais para documentação, análise e teste do código. Elas são, em sua maioria, específicas da plataforma Windows [31] ou mesmo não possuem interface gráfica [35]. Algumas destas principais ferramentas serão discutidas no decorrer deste documento. Essa falta de ferramentas faz com que um desenvolvedor iniciante de aplicações de bancos de dados reserve um grande intervalo de tempo para compreender a estrutura do sistema, para identificar quais são as tabelas existentes e para entender a semântica dessas tabelas, dos procedimentos, dos objetos do projeto e suas interdependências. Isto ocorre porque não existem metodologias baseadas em padrões de projeto a serem seguidas nem ferramentas que satisfaçam as necessidades de análise, refatoramento, testes e documentação de forma independente de plataforma e livre de custos de licença.

Por não possuir ferramentas consolidadas no mercado que auxiliem o desenvolvedor durante as diferentes fases de desenvolvimento do código PL/SQL, muitas destas etapas não recebem atenção especial dos desenvolvedores. Como consequência disso, tem-se código não-padronizado, desprovido de testes que assegurem sua corretude, sem documentação acessível por navegadores ou outros visualizadores e sem análise e métricas sobre a produtividade do projeto.

Este conjunto de dificuldades encontradas serviu de motivação para o desenvolvimento de uma ferramenta que atenda a essas necessidades da etapa de pós-codificação. O diferencial do sistema proposto por este trabalho consiste na provisão das seguintes funcionalidades: documentação automática, geração de gráficos de dependências, renomeação de entidades do banco de dados e geração automática de uma fachada para os pacotes de dados, além de possuir código aberto, ser livre de custos de licença e independente de sistema operacional.

**1.2 Objetivos**

O objetivo geral deste trabalho é a especificação e implementação de uma ferramenta que auxilie o desenvolvedor de código PL/SQL em algumas etapas do ciclo de desenvolvimento de software, levando em consideração as suas principais dificuldades e necessidades. O sistema proverá funcionalidades para as etapas de documentação, análise, refatoramento e testes de código, para auxiliar no desenvolvimento de aplicações de bancos de dados.

Para tal, é necessária a catalogação de padrões de projetos para ambientes de desenvolvimento de aplicações de BD, a fim de que os mesmos possam ser utilizados para o desenvolvimento de novas ferramentas de programação em PL/SQL.

Quanto aos objetivos específicos pode-se citar a realização das seguintes atividades:

• Estudo detalhado de diversos padrões de projeto existentes e de ambientes de desenvolvimento de aplicações de BD a fim de verificar suas principais funcionalidades e arquiteturas;

• Definição de um conjunto de funcionalidades para ambientes de programação em PL/SQL;

• Definição de um sistema computacional para a manutenção de códigos PL/SQL;

• Implementação de um protótipo para validar algumas funcionalidades previstas para o sistema proposto

• Desenvolvimento de um estudo de caso com base no protótipo implementado e em uma aplicação de BD para validação dos padrões de projeto propostos.

**1.3 Estrutura do documento**

Este documento está organizado da seguinte forma. O segundo capítulo contém a fundamentação teórica básica para que o leitor se familiarize com os principais conceitos usados na descrição do trabalho proposto neste documento e que eventualmente serão citados no decorrer do texto.

O capítulo três mostra a especificação da ferramenta PL/SQL Helper, indicando as principais funcionalidades deste sistema e as justificativas das escolhas de projeto. Este capítulo também mostra como a ferramenta PL/SQL Helper foi especificada. No quarto capítulo, será apresentado o protótipo do sistema proposto. O quinto capítulo exibe o estudo de caso utilizado para validação da ferramenta desenvolvida, junto com os principais resultados obtidos à ilustração de seu uso.

Finalmente, no sexto capítulo, serão apresentadas as conclusões a respeito do trabalho realizado, incluindo uma lista das principais contribuições para a área e uma indicação dos trabalhos futuros que deverão ser desenvolvidos para aperfeiçoamento da ferramenta proposta.

**2 Fundamentação Teórica**

Este capítulo lista os conceitos envolvidos neste trabalho e que são pré-requisitos para o bom entendimento deste documento que o descreve. Ele exibe um breve resumo sobre a história do desenvolvimento de SQL e em seguida, define cada uma das técnicas de engenharia de software abordadas por este trabalho.

**2.1 Introdução**

A linguagem *Structured Query Language* (SQL) teve origem em um projeto da IBM no início dos anos 1970 para se desenvolver um modelo de dados para Sistemas de Gerenciamento de Bancos de Dados (SGBD) relacional [11]. Este modelo foi proposto para substituir o até então usado, modelo de dados hierárquico [48].

O projeto da IBM, chamado de *System R,* possuía umalinguagem de consulta denominada *Sequel* (S*tructured English Query Language*) que foi abreviada para SQL (em anos posteriores) e era composto de várias fases, sendo concluído apenas em 1979, cinco anos após seu início.

SQL sofreu diversas alterações até ser padronizada em 1986 pela *American National Standards Institute* (ANSI) como SQL-86, e posteriormente ganhou certificação ISO (*Internation Standards Organization*) [12]. Desde então, SQL tem sido a linguagem padrão para manipulação e definição de dados, seja em aplicativos pequenos destinados a PDA e Palm como em consultas complexas envolvendo grandes massas de dados mantidas em mainframes e servidores. Suas principais vantagens em relação às abordagens anteriores são a fundamentação matemática de corretude baseada na álgebra relacional, a abstração da ordem em que os processos são executados e de como os dados são obtidos pelo SGBD e a sua sintaxe declarativa e de fácil compreensão, mesmo para os leigos.

A linguagem PL/SQL foi desenvolvida pela Oracle em 1991 sendo sua versão 1.0 somente lançada em 1992 como uma extensão opcional do, até então atual, Oracle 6. Baseada em Ada (uma variante de Pascal que foi usada pelo Departamento de Defesa Americano), PL/SQL é procedural e estende SQL [13]. A principal idéia da Oracle em torno desta linguagem era trazer características robustas das linguagens estruturadas tais como tratamento de exceções, laços, coleções e resolução de eventos (denominados gatilhos) para o ambiente de desenvolvimento em bancos de dados. A linguagem fez sucesso e já na sua segunda versão, eram permitidas o uso de *stored procedures*, *functions*, *packages*, tipos definidos pelo usuário, *PL/SQL tables,* além de várias extensões de pacotes incluindo *DBMS\_OUTPUT* e *DBMS\_PIPE* [13]. A união de SQL com o PL/SQL da Oracle proveu aos desenvolvedores, um amplo poder sobre os dados, pois integra a fácil manipulação e definição de SQL com o alto poder de processamento de PL/SQL. A partir daí, as principais ferramentas de gerenciamento de bancos de dados passaram a disponibilizar uma linguagem procedural própria para o processamento interno de dados.

**2.2 Conceitos Básicos**

Esta seção contém as definições de alguns termos usuais da área de bancos de dados, utilizados neste documento. Estes conceitos são fundamentais para compreensão das funcionalidades da ferramenta proposta.

**2.2.1 Rotina**

Rotina é um conceito utilizado para definir um trecho de código que é executado repetidamente. Um bloco de código que sempre é executado para um determinado fim é chamado de rotina. Rotinas podem ser classificadas como procedimentos ou funções, de acordo com suas características. Idealmente, uma função é uma rotina que pode retornar ao invocador, algum valor resultante de seu processamento e pode fazer parte de expressões, enquanto que um procedimento é apenas chamado como uma instrução da linguagem e não possui um valor de retorno. Em PL/SQL, porém, é possível retornar valores tanto por meio de funções como por procedimentos, pois existe a noção de parâmetros de saída que são parâmetros que são passados por referência e podem ser atualizados pela rotina.

**2.2.2 Pacotes**

Pacote é um recurso que foi incorporado ao PL/SQL com base nos pacotes da linguagem Ada. O pacote é uma entidade que permite que objetos, que se relacionam logicamente, sejam armazenados juntos. O pacote Oracle tem duas partes separadas: a especificação ou declaração do pacote e o corpo do pacote. Na especificação, são mantidos os atributos globais do pacote e as declarações de suas rotinas. No corpo, são guardadas as implementações das rotinas e as declarações dos atributos locais ao corpo do pacote. A principal vantagem do pacote é que seus atributos e suas rotinas são acessíveis externamente, podendo ser invocados por outras rotinas. [27]

**2.2.3 Pós-codificação**

As atividades realizadas na etapa de pós-codificação são todas as ações executadas pelo desenvolvedor para garantir a qualidade do *software,* após ele ter sido codificado. Para isso, ele pode utilizar ferramentas de análise, reformatação, documentação, refatoramento, dentre outras. A idéia é trabalhar após a geração do código (código já escrito) para o seu aperfeiçoamento e garantia de alguns pontos importantes como legibilidade, manutenabilidade, robustez e documentação.

**2.2.4 Métricas**

Métricas são atributos que visam quantificar certos quesitos observados para garantir a qualidade do software. Quantidade de linhas em uma rotina, número total de pacotes, total de procedimentos e funções, e quantidade de funções dependentes, são exemplos de métricas.

**2.3 Técnicas de Engenharia de Software**

Esta seção descreve brevemente as principais técnicas de engenharia de software utilizadas pela ferramenta proposta neste documento. Estas técnicas foram escolhidas com base no estudo realizado sobre os trabalhos correlatos, cujos resultados serão exibidos na seção 2.5. Este estudo evidenciou a falta de ferramentas que ofereçam recursos de manutenção de projetos PL/SQL aos desenvolvedores da da área de bancos de dados, motivando então, a incorporação das mesmas em nossa ferramenta.

**2.3.1 Padrões de Projeto**

Padrões de projeto são soluções para problemas comuns que podem ocorrer em ambientes diferentes, mas que podem ser abordados usando-se a mesma estratégia. Existem padrões de desenvolvimento para muitos problemas de software como instanciação de objetos, construtores de objetos, gerenciamento de eventos, monitoramento de comportamentos, dentre outros. Vimos que alguns desses padrões possuem implementações em PL/SQL, como o *Decorator*, *Template*, *Observer* e *Dependency Injection* [14, 15, 16].

Para o presente trabalho, foi estudado um padrão de projeto de software para permitir sua adaptação ao ambiente de bancos de dados. Como pré-requisitos para esta escolha, foram verificadas adaptabilidade e disponibilidade. O primeiro quesito baseou-se em padrões que fossem úteis para o ambiente de bancos de dados, enquanto que para o segundo quesito, foi analisado se para tal padrão já havia implementação em PL/SQL na literatura. O padrão *Facade* [18] foi escolhido por representar um problema real, ser fácil de ser identificado e por satisfazer os dois quesitos listados anteriormente.

O padrão *Facade* [18] tem como idéia principal, criar uma interface de comunicação entre o código escrito e as entidades externas que podem usar as funcionalidades deste código. É uma camada intermediária entre as camadas lógica e de interface que aumenta a modularidade do software. A fachada centraliza ações de vários componentes lógicos da camada inferior. As entidades de nível de interface podem fazer uso destas ações através desta fachada de modo que elas não precisem saber em qual componente se encontra a implementação de determinada funcionalidade. O padrão fachada também provê um encapsulamento das chamadas inferiores, produzindo uma vantagem semelhante ao padrão *Template*. Este padrão especifica, através de um contrato, quais parâmetros são necessários para cada ação e qual o retorno ou a consequência destas ações. Uma vez especificado esse contrato, quaisquer modificações no código da funcionalidade não acarretam em alterações nas entidades que as utilizam, desde que não sejam quebrados os contratos definidos. Logo, uma mudança na ordem dos parâmetros ou na declaração da funcionalidade, por exemplo, não acarretam necessariamente em uma mudança no componente da camada de interface. A figura 2.1 apresenta um diagrama de componentes UML para ilustrar a presença de uma fachada na arquitetura de um sistema de software modularizado em camadas. A fachada permite que seja minimizada a dependência de código, uma vez que os componentes lógicos não são chamados explicitamente pelas entidades de interfaces gráficas.

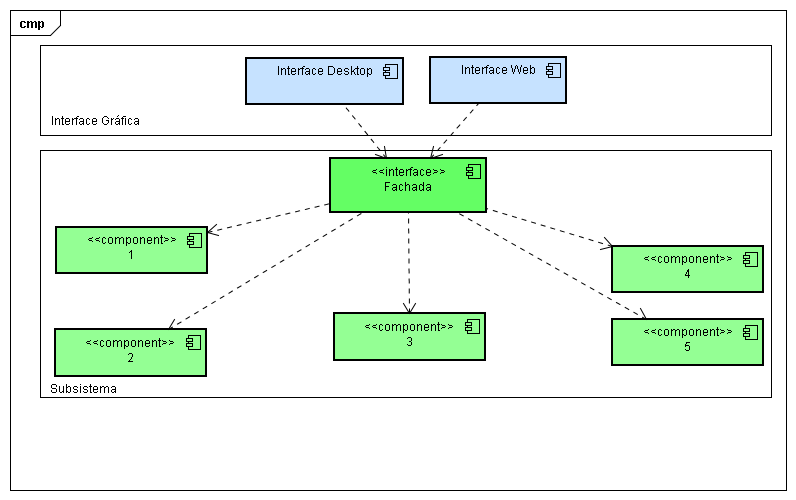


Figura 2.1 - Arquitetura de software com fachada

**2.3.2 Refatoramento**

Refatoramento, do inglês *Refactoring,* é uma técnica que visa modificar a estrutura interna de um código existente sem alterar seu comportamento externo [19]. Seu fundamento se baseia em uma série de transformações que não impliquem em modificações no comportamento do código. Cada transformação está geralmente associada a uma parte do código, mas o conjunto de transformações feitas pode acarretar em reestruturações significantes no código. Tendo em vista que cada modificação é pequena, então elas são menos suscetíveis a falhas e mais fáceis de verificar a corretude [19]. Existem diversos tipos de refatoramento de códigos como extração de métodos, extração de classes, encapsulamento de atributos, renomeação de entidades, dentre outros. A figura 2.2 exibe um exemplo de refatoramento realizado sobre um código, visando extrair um método que está codificado dentro de outro método.

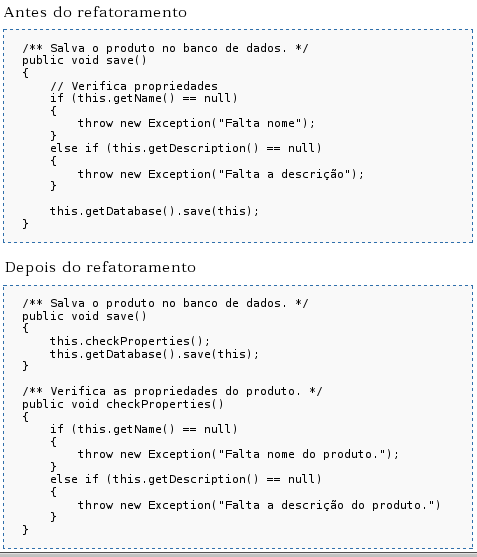


Figura 2.2 – Exemplo de refatoramento

Na figura 2.2, é possível notar que a extração realizada pelo refatoramento aumentou a facilidade de leitura do código do método “*save*”. Esse método englobava todo o processamento para verificar as propriedades do produto a ser salvo na base de dados, dificultando a sua compreensão. Após a extração, o método apenas centraliza chamadas às funções auxiliares que implementam as ações intermediárias.

Para o presente trabalho, foi estudado um refatoramento bastante comum no desenvolvimento de software que é o renomeamento de entidades. A escolha foi feita com base nos refatoramentos definidos por Martin Fowler [19]. Martin especificou os tipos de refatoramentos eventualmente necessários em um projeto de software. O quesito de adaptabilidade foi o principal responsável pela escolha deste refatoramento, visto que os principais tipos de refatoramentos existentes são específicos para linguagens orientadas à objetos, como encapsulamento de atributos e extração de sub e super classes.

O refatoramento escolhido tem como objetivo ajudar o programador que deseja alterar o nome ou rótulo de alguma entidade interna de seu projeto de banco de dados. Normalmente, esse nome é utilizado por todos os outros componentes do sistema para acessar as funcionalidades dessa entidade ou mesmo ler o conteúdo de um de seus campos. Logo, uma modificação nesse rótulo, faria com que todas as outras entidades falhassem. Uma vez modificado o rótulo do objeto, o programador necessita alterar em todas as partes do código, as chamadas explícitas a esse rótulo, atualizando-as para o novo valor. O refatoramento para renomear entidades automatiza o processo de busca e modificação dessas chamadas ao rótulo antigo, garantindo que nenhuma entidade falhe após a atualização. Esse caso de refatoramento é bastante aplicável ao ambiente de bancos de dados, visto que em um pacote, podem ser definidos atributos de acesso global e local, tendo a necessidade de uma busca automática para possível modificação desses atributos, sem adicionar riscos de violação de integridade entre outras entidades do sistema.

**2.3.3 Teste de Software**

Teste de software é uma técnica de investigação do software com o objetivo de identificar a qualidade e corretude do sistema desenvolvido, levando em consideração o ambiente, as entidades envolvidas e as condições em que ele será aplicado [25]. Essa etapa do ciclo de desenvolvimento tenta minimizar o número de falhas que podem ocorrer em um software, para que ele permaneça estável durante a maior parte de seu tempo de funcionamento. Erros são, por enquanto, inevitáveis. Mesmo os maiores e mais robustos sistemas de software apresentam falhas e estão em constante aperfeiçoamento [21].

Existem diversos fatores que contribuem para a dificuldade de proteger completamente um software de falhas. Os sistemas, em sua maioria, são complexos e possuem muitos estados. Cada um desses estados possuem fórmulas, processamentos e/ou algoritmos que juntos com uma grande quantidade de pessoas envolvidas no projeto, aumentam ainda mais a complexidade do sistema como um todo. Existem ainda fatores que contribuem para a inserção de erros dentro do projeto, como mudança nos requisitos do sistema, problemas na comunicação interna da equipe de desenvolvimento, erros de programação, pressão devido aos cronogramas, dentre outros.

Para ilustrar a dificuldade em se criar um software completamente livre de falhas, uma taxa de 15% à 30% de defeitos a cada mil linhas de código é considerada aceitável para a indústria de tecnologia da informação[22][23]. Portanto, a etapa de testes foi aos poucos ganhando mais espaço no ciclo de desenvolvimento de software. Atualmente, a etapa de testes de software chega a representar cinquenta por cento do tempo de desenvolvimento de um sistema [24]. Esse tempo gasto com testes exaustivos tem o propósito de aumentar a qualidade final do produto. Qualidade que, segundo Hetzel [25], possui três dimensões, cada uma delas constituída de componentes com menor nível de detalhe. O quadro 2.1 mostra as três dimensões da qualidade de software definidas por Hetzel.

Quadro 2.1 - Dimensões de qualidade de software

|  |  |  |
| --- | --- | --- |
| **Funcionalidade** | **Engenharia** | **Adaptabilidade** |
| Corretude | Eficiência | Flexibilidade |
| Confiabilidade | Testabilidade | Reusabilidade |
| Usabilidade | Documentação | Manutenção |
| Integridade | Estrutura |  |

Um bom processo de testes provê medidas para todos os componentes das diferentes dimensões da qualidade do software, mas como a importância de cada entidade varia de aplicação para aplicação, então existem diferentes testes para cada um dos fatores envolvidos na qualidade do sistema. Para cada teste, são criados casos de testes. Os casos de testes são configurações especiais feitas pelo desenvolvedor para encontrar falhas no produto e eles definem as entradas e quais saídas são esperadas para elas. Segundo Myers [26], um bom caso de teste é aquele com alta probabilidade de encontrar uma falha ainda não descoberta, enquanto um caso de teste bem sucedido é aquele que conseguiu descobrir um erro até então desconhecido. Com esse ponto de vista, percebe-se que enquanto não são encontradas falhas no sistema, seu processo de testes está falhando, logo é difícil estimar quando se deve terminar a execução de testes no produto. De acordo com Pan [24], a etapa de testes é potencialmente infinita, não sendo possível estimar o ponto de parada. Mas ainda segundo ele, é possível fazer um balanceamento entre a qualidade do produto desejada e o tempo de desenvolvimento do software, atingindo um nível satisfatório de qualidade dentro de um tempo disponível.

Existem níveis diferentes de testes que servem para testar comportamentos distintos de software. Podemos citar testes unitários, testes de integração [49] e testes de sistema [50]. Testes unitários (ou testes de unidade) são testes que visam encontrar falhas em entidades pontuais do projeto, ou seja, sem levar em consideração seus relacionamentos externos. A idéia principal do teste de unidade é garantir o funcionamento de todas as pequenas partes do sistema de forma independente para que as alterações futuras em seus relacionamentos externos não acarretem em inserções de falhas nos módulos isolados.

O teste de integração é uma técnica sistemática para a construção da estrutura do programa, realizando testes para descobrir erros associados à interface. Este tipo de teste resolve erros da etapa de integração do software, que são os erros provenientes das relações entre os objetos previamente testados unitariamente.

Teste de sistema refere-se ao teste da aplicação como um todo, para averiguar se esta provê o comportamento esperado. Este tipo de teste também é útil para validação das funcionalidades do sistema.

**2.3.4 Documentação de Software**

Documentação de software é uma atividade da engenharia de software que visa gerar documentos a respeito da aplicação a ser desenvolvida. Segundo Sommerville [28], os documentos produzidos pela tarefa de documentação de software podem ser tratados como: meios de comunicação entre as diversas etapas de desenvolvimento; repositórios de informação que podem ser usados por equipes de manutenção; meios provedores de informações gerenciais que ajudam no planejamento do projeto; manuais de utilização do sistema para usuários finais. Para satisfazer todas essas finalidades, é necessário criar tipos diferentes de documentos. Não é possível definir um conjunto específico de documentos requeridos em um projeto de software. Isto varia de acordo com a equipe de desenvolvimento, o tamanho e a cultura da empresa desenvolvedora. Porém, ainda segundo Sommerville, é possível verificar a existência de duas classes distintas de documentação: a documentação de processo e a documentação de produto.

A documentação de processo gera um documento interno, usado pela equipe e pela empresa de desenvolvimento e tem como objetivo registrar o processo de desenvolvimento e a manutenção do software. Essa documentação também guarda os principais planos, estimativas, cronogramas, métricas de qualidade e padrões adotados durante a etapa de desenvolvimento do produto. A principal característica da documentação de processo é que seu documento se torna desatualizado rapidamente. Os planejamentos contidos na documentação podem ser reformulados semanalmente ou mensalmente, por exemplo, e tornam a atualização do documento uma atividade constante.

A documentação de produto está associada com o produto de software a ser entregue. Diferente dos documentos de processo, os documentos de produto possuem uma vida longa. Essa documentação contém informações que ensinam aos usuários, como usar o produto final além de servir como documentação para a equipe de manutenção. Por este motivo, Sommerville divide a documentação de produto em documentação de usuário e de sistema.

Com os documentos produzidos por essa atividade da engenharia de software, é possível minimizar o tempo de aprendizado da equipe de manutenção do software, isolar falhas de componentes do sistema, e prover ao usuário final, um meio seguro e rápido para aprender a melhor maneira de utilizar o produto, além de servir como dados históricos para a equipe e a empresa de desenvolvimento.

**2.4 Ambientes PL/SQL**

Esta seção descreve o *SQL* Developer [29] da Oracle e o PL/SQL Developer [30] da Allround Automations. As duas empresas são bem consolidadas no mercado e seus produtos são bastante difundidos, e por isso, uma análise destes ambientes compõe o trabalho descrito neste documento.

O Oracle SQL Developer [29] é um produto sem custos de uso e multi-plataforma. A ferramenta pode ser utilizada inicialmente por usuários Linux, Windows ou Apple, mas essa lista de sistemas operacionais pode vir a aumentar, visto que o produto é desenvolvido sobre a plataforma Java. Portanto, qualquer sistema operacional com uma máquina virtual Java instalada é capaz de executar o programa de desenvolvimento PL/SQL da Oracle. A ferramenta é capaz de editar códigos PL/SQL e consultas SQL. Além disso, ela possui um subsistema de depuração, a partir do qual é possível navegar pela execução de uma rotina para verificar falhas. O programa exporta dados de consultas para formatos conhecidos como XLS, HTML, XML e CSV, além de permitir o uso de controle de versões.

O PL/SQL Developer [30] é uma ferramenta paga e desenvolvida especificamente para plataforma Windows. Esta ferramenta edita códigos assim como a ferramenta da Oracle descrita acima e também é capaz de realizar depurações em rotinas. Uma vantagem do PL/SQL Developer em relação ao ambiente descrito anteriormente é a disponibilização de uma janela de comandos que simula o ambiente SQL\*PLUS do SGBD Oracle, tornando possível a execução e interpretação de *scripts* dentro do mesmo ambiente de desenvolvimento. Outra vantagem em relação ao produto da Oracle é a possibilidade de gerar relatórios como consultas, comparações de usuários e relatórios personalizados. Uma funcionalidade para a fase de pós-codificação é a janela de diagrama na qual é possível ver uma representação visual das tabelas do sistema e seus relacionamentos.

Como foi observado, estes ambientes PL/SQL mais difundidos no mercado satisfazem as necessidades de codificação dos desenvolvedores PL/SQL. Estes ambientes, porém, não atendem às demandas associadas às etapas de pós-codificação, como análise, documentação, métricas, refatoramento e testes, tendo o PL/SQL Developer uma única funcionalidade para este grupo de atividades.

**2.5 Trabalhos Correlatos**

Como a ferramenta proposta por este trabalho se trata de uma ferramenta para auxiliar projetos PL/SQL já desenvolvidos (ferramenta de pós-codificação), não foram considerados como trabalhos correlatos, os editores de PL/SQL Developer e SQL Developer, pois como visto anteriormente, eles não possuem funcionalidades para estas atividades. Análise, manutenção, documentação de código, refatoramento e testes não fazem parte de suas principais funcionalidades, pois suas maiores contribuições são para o nível de codificação como: iluminação de sintaxe, complementação de código (nome de variáveis e procedimentos), depuração, salientação de erros de compilação, entre outras. Foram pesquisados, no mercado e na academia, quatro programas de atividades de pós-codificação, levando em consideração os módulos de documentação, análise, teste e refatoramento: ClearSQL [31], TOAD Soft [32], PLDOC [35] e utPLSQL [2]. Os programas ClearSQL e TOAD Soft foram escolhidos por sua difusão no mercado e por possuírem diversas funcionalidades, enquanto o PLDOC e o utPLSQL foram escolhidos por contemplarem recursos de geração de documentação e testes, respectivamente, que pertencem à etapa de pós-codificação.

*ClearSQL* [31] se entitula como um revisor de código e uma ferramenta de controle de qualidade de código PL/SQL para Oracle, dando recomendações para que o desenvolvedor siga padrões de escrita de código como nomeação de variáveis, indentação e comentários. Dentre suas principais funcionalidades estão: (1) a formatação de código: que visa controlar indentação, linhas de espaço em branco, aplicação de palavras-chave à identificadores; (2) o gráfico de fluxo do código: que exibe uma representação visual do código da função; e (3) o uso de métricas: que informam o número de linhas de código, linhas de comentários, linhas em branco, entre outros. Os principais problemas encontrados no sistema são: (1) Ele foi unicamente desenvolvido para ambiente Windows; (2) Software é pago, sendo necessário portanto, a realização de um investimento inicial para fazer uso de suas funcionalidades e (3) Não possui funcionalidade para execução de testes unitários, documentação ou refatoramentos.

*TOAD* Soft (*Tool for Application Developers*) [32] é um software desenvolvido pela empresa Quest Soft e é bastante difundido na área de bancos de dados para fins de gerenciamento de banco de dados. A ferramenta consiste numa grande aplicação que abrange funcionalidades desde edição de código (como PL/SQL Developer) até alguns exercícios de pós-codificação. Essa segunda parte do sistema é o alvo de nossa análise, e é detalhada a seguir.

TOAD Soft é capaz de formatar códigos PL/SQL e exibir métricas de maneira semelhante ao ClearSQL, mas ao contrário do seu concorrente, ele possui a funcionalidade de planejar e realizar testes unitários em funções e procedimentos definidos pelo usuário, e também exibe o diagrama visual das tabelas (semelhante ao PL/SQL Developer), além de possuir um otimizador de código. Porém, a principal desvantagem do TOAD é que ele é um software específico para plataforma Windows. Apesar de possuir versões para Oracle, MySQL e Sql Server, cada uma destas versões possui uma compilação diferente, possuindo diferentes características e funcionalidades. Até o momento de escrita deste documento, as versões do TOAD para os sistemas MySQL e Sql Server ainda estavam em fase *Beta* sem as principais funcionalidades de otimização de desempenho e análise, enquanto que a compilação Oracle já está na versão 9.7. Além disso, o produto da empresa Quest Software não oferece recursos para geração de documentação Web nem ferramentas de refatoramento.

PLDOC [35] é uma ferramenta de código aberto e em fase de construção, que provê a geração automática de documentação para código PL/SQL no padrão Javadoc [34]. A ferramenta não possui um aplicativo de interface gráfica, consequentemente suas funcionalidades são acessadas via linha de comando, e todas as tarefas de documentação são configuradas manualmente pelo usuário todas as vezes que a ferramenta é invocada. O PLDOC, até o momento de escrita deste documento, possui apenas a funcionalidade de documentação para pacotes, enquanto suas demais funcionalidades estão ainda em fase desenvolvimento. Uma vantagem desta ferramenta é a portabilidade. O PLDOC foi desenvolvido em Java, portanto pode ser executado em qualquer plataforma que possua uma máquina virtual Java instalada. Sua principal desvantagem é não prover funcionalidades para os outros módulos da etapa de pós-codificação, como análise, teste e refatoramento.

UtPLSQL [2] é um *framework* de testes unitários para desenvolvedores da linguagem PL/SQL. Desenvolvida por Steven Feuerstein, a ferramenta é capaz de realizar testes automáticos de pacotes, funções e procedimentos de uma aplicação de banco de dados. O utPLSQL [2] é independente de plataforma e é instalado através de *scripts* SQL que precisam ser configurados manualmente pelo usuário. Os casos de teste são criados definindo-se pacotes especiais para testes e então, codificando as verificações de teste nas funções deste pacote.

As principais dificuldades do utPLSQL são: sua instalação, pois são necessários privilégios de acesso ao banco de dados, juntamente com a definição de uma conta de usuário exclusiva para a execução do *framework*; e a falta de interface gráfica para utilização da ferramenta.

**2.6 Conclusão**

Foi observado que os principais ambientes para desenvolvedores PL/SQL atendem apenas, em sua maioria, aos requisitos de codificação. Foi constatado, porém, que algumas funcionalidades relativas à etapa de pós–codificação podem ser encontradas no mercado como, documentação automática, métricas, diagrama de tabelas e testes unitários. Para métricas e diagrama de tabelas, a maior desvantagem é o fato de suas ferramentas possuírem custos de licença e serem específicas para plataforma Windows. Já a documentação e os testes unitários são implementados por sistemas sem interface gráfica, dificultando a acessibilidade e tornando manual o processo de utilização de suas funcionalidades. Além dessas dificuldades, outras atividades como gráficos de dependências, renomeação de entidades e criação automática de pacotes fachada não são contempladas pelas ferramentas analisadas.

**3 Especificação da Ferramenta PL/SQL Helper**

Este capítulo contém a especificação da ferramenta desenvolvida por este trabalho. Inicialmente, uma introdução exibirá o porquê de se especificar uma nova ferramenta para as atividades da etapa de pós-codificação, e em seguida será demonstrada a arquitetura do PL/SQL Helper e como ela foi modelada. Na terceira seção, serão exibidas as funcionalidades da ferramenta proposta, e então será mostrado um comparativo entre as ferramentas analisadas neste documento e o sistema PL/SQL Helper. Por fim, serão dadas conclusões a respeito do capítulo.

**3.1 Introdução**

Existem diversos editores de texto para códigos PL/SQL [29, 30, 36]. Eles possuem salientação de palavras reservadas, indentação de código e alguns deles têm formatações de texto [29,30]. Porém, poucas ferramentas de análise, manutenção e testes de pós-codificação são oferecidas aos desenvolvedores da área de bancos de dados. É desconhecida, até o momento, a existência de uma ferramenta que auxilie o desenvolvedor com testes, documentação, análise e refatoramento de um projeto de bancos de dados. É importante que um desenvolvedor de códigos PL/SQL use uma ferramenta que analise as dependências de suas rotinas e exiba um diagrama visual mostrando quais componentes podem ter seu comportamento afetado por outros. Outras funcionalidades importantes incluem a geração de uma documentação HTML com *links* de fácil navegação sobre seus componentes e o uso de padrões de projeto em seu código. Não se tem conhecimento de uma ferramenta do mercado ou da academia que proveja essas soluções de forma independente de sistema operacional e livre de custos de licença.

**3.2 A arquitetura do PL/SQL Helper**

PL/SQL Helper é uma ferramenta de código aberto, independente de sistema operacional e desenvolvida em camadas, segundo o padrão de projeto Layers [44]. Esta divisão tem como objetivo, desacoplar as entidades de níveis diferentes do projeto de software. Foram criadas quatro camadas para a arquitetura do PL/SQL Helper: uma camada de dados, uma camada de acesso aos dados, uma camada lógica, que contém as entidades responsáveis por aplicar as regras de negócio da ferramenta, e uma camada de interface gráfica, que é responsável por prover ao usuário, um meio de utilizar as funcionalidades providas pela ferramenta aqui proposta. Estas camadas podem ser visualizadas na figura 3.1, que exibe um modelo de componentes UML para ilustrar a arquitetura do PL/SQL Helper.

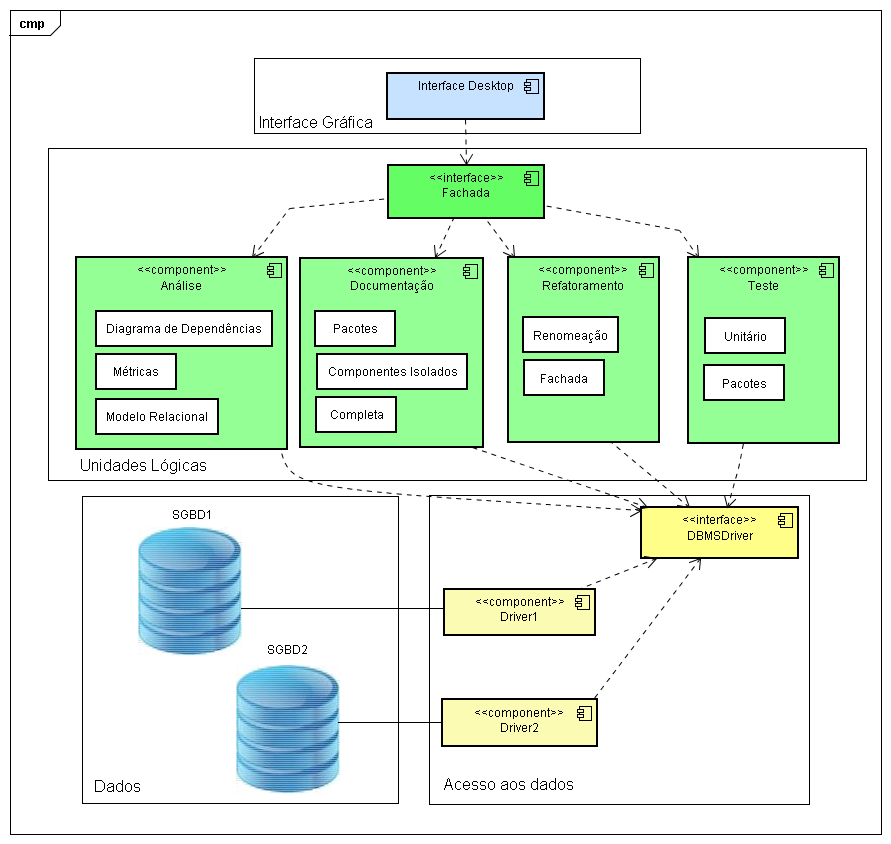


Figura 3.1 - Arquitetura da ferramenta proposta usando componentes UML

A camada de dados é representada pelos sistemas de gerenciamento de bancos de dados, como Oracle por exemplo. Para realizar o acesso a esses dados, uma camada foi criada contendo entidades chamadas *drivers.* Estas entidades estendem um conjunto de funcionalidades definidas no componente de interface da camada de acesso aos dados, chamado DBMSDriver. Os *drivers* implementam estas funcionalidades de acordo com o SGBD que lhe é designado. Para cada SGBD, um *driver* é criado, dando à ferramenta uma independência de sistema de gerenciamento de bancos de dados.

A camada de unidades lógicas é responsável por prover as funcionalidades do PL/SQL Helper, pois ela contém os componentes que realizam cada uma das atividades propostas. Os componentes desta camada consultam informações a respeito do banco de dados através da interface DBMSDriver. Ao todo, são quatro componentes: análise, documentação, refatoramento e teste.

Usando o padrão de projeto *Facade* [18]*,* explicado na seção 2.3.1, foi criado um componente que comunica a camada de unidades lógicas com a camada de interface gráfica da ferramenta. Esse componente é responsável por centralizar as operações do sistema para o acesso do componente gráfico da ferramenta.

Na camada superior, foi criada a GUI (*Graphical User Interface)* que tornará visivelmente atrativas, as funcionalidades do sistema. O usuário poderá utilizar tabelas, janelas, botões e outros recursos gráficos para atingir mais facilmente seus objetivos. A GUI poderá ser modificada sem a necessidade de alteração na parte lógica do código, visto que o sistema está organizado em camadas de acordo com padrão de camadas de engenharia de software [44].

**3.3 As funcionalidades do PL/SQL Helper**

As funcionalidades oferecidas pela ferramenta proposta por este trabalho foram levantadas com base nas demandas de pós-codificação elicitadas na seção de motivação (seção 1.1). Também foram consideradas as funcionalidades oferecidas pelas ferramentas analisadas na seção 2.5. Ao todo, foram especificadas dez funcionalidades divididas em quatro módulos distintos: análise, documentação, refatoramento e teste. Três destas dez funcionalidades não estão presentes nas ferramentas analisadas (geração de fachada, gráfico de dependência e renomeação de entidades), enquanto as outras sete estão presentes, porém possuem desvantagens em relação ao PL/SQL Helper, como visto no estudo dos trabalhos correlatos (seção 2.5). O modelo UML, que pode ser visualizado no Anexo 1 deste documento, exibe o diagrama de classes do PL/SQL Helper e contém as funcionalidades especificadas neste capítulo, juntamente com detalhes dos componentes definidos para a arquitetura da ferramenta.

**3.3.1 Documentação**

Como visto na seção 3.1, a fácil utilização da documentação é importante para uma aplicação de banco de dados. Por este motivo, foi decidido fazer uso do padrão Java Doc [34]. A seção 2.5 apresentou o PLDOC [35] como sendo uma ferramenta de código aberto e que faz uso do padrão Java Doc para gerar a documentação da aplicação de banco de dados desejada, mas que não possui interface gráfica. Por isso, o PLDOC foi incorporado à ferramenta proposta por este trabalho, para atender as funcionalidades especificadas para seu módulo de documentação, e ao mesmo tempo representar uma interface gráfica para o PLDOC.

Como a busca era por uma ferramenta para documentar um sistema já codificado, então outras ferramentas com objetivo de facilitar o processo de documentação durante a fase de desenvolvimento foram excluídas da pesquisa. O objetivo principal deste módulo de documentação é aproveitar os comentários escritos nos códigos pelos desenvolvedores e agrupá-los de maneira lógica e de fácil acesso.

Adaptações precisaram ser feitas para incorporar o PLDOC ao PL/SQL Helper. O *framework* está em fase de desenvolvimento e, por isso, possui apenas documentação de pacotes. Sua versão mais recente, utilizada pela ferramenta proposta por este trabalho, é a 0.8.3. Nesta versão, a funcionalidade oferecida está limitada a gerar documentação para arquivos de extensão *SQL* dentro de uma pasta especificada. Para que sua funcionalidade seja utilizada, o usuário deve copiar os códigos dos pacotes de dados, transformá-los em arquivos, colocar estes arquivos em uma única pasta e invocar o *framework* através de linha de comando. O PL/SQL Helper automatizou todo este processo, selecionando em tempo real os pacotes do usuário, criando os arquivos necessários e invocando o PLDOC.

Para a ferramenta proposta por este trabalho, foram definidas as seguintes funcionalidades para o módulo de documentação: documentação de pacotes isolados, documentação de um conjunto de pacotes, documentação de entidades isoladas, documentação de um conjunto de entidades e documentação completa de todas as entidades e pacotes. Esses tipos básicos podem ser agrupados em três classes que são oferecidas pelo PL/SQL Helper: documentação de pacotes, documentação de entidades e documentação completa. A primeira classe é responsável pela documentação dos diversos pacotes de dados criados na conta do usuário de banco de dados, enquanto a segunda classe é responsável por documentar os elementos criados fora de pacotes de agrupamentos lógicos e a terceira classe documenta o projeto como um todo. Essa separação em classes de funcionalidades é bastante útil para aplicações ainda em desenvolvimento ou em crescimento, pois ela permite documentar os novos módulos em separado, visto que outras partes do sistema já supostamente possuem uma documentação gerada. A Figura 3.2 exibe, através de um diagrama de classes UML, a definição da entidade de documentação da arquitetura do PL/SQL Helper.

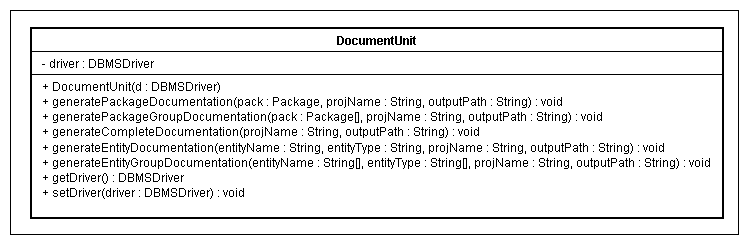


Figura 3.2 - Definição da entidade de documentação do PL/SQL Helper

O módulo de documentação do PL/SQL Helper permite que o usuário selecione visualmente quais entidades ou pacotes do seu banco de dados são analisados para documentação. Será possível escolher uma ou mais entidades ou pacotes, o título do sistema de software a ser documentado e em qual diretório serão gravados os arquivos de documentação gerados. Com essas escolhas, a ferramenta executa chamadas automáticas ao PLDOC interno. Logo após, uma mensagem de sucesso será exibida e o usuário poderá acessar, na pasta definida por ele, os arquivos que contêm a documentação gerada pelo PL/SQL Helper.

**3.3.2 Análise**

O módulo de análise provê funcionalidades que auxiliam na manutenção de código. Como visto anteriormente, a única funcionalidade de análise para código PL/SQL que se tem conhecimento é o sistema de métricas do Toad Soft [32]. Para a ferramenta proposta por esse trabalho, foram definidas três funcionalidades analíticas, que foram escolhidas com base nas atuais funcionalidades oferecidas pelas ferramentas analisadas na seção 2.5 e com as demandas elicitadas na seção 1.1. Para análise de códigos PL/SQL foram definidas: métricas e gráfico de dependência de rotinas. Para análise do banco de dados, foi selecionada a exibição do modelo relacional das tabelas. A Figura 3.3 exibe a definição, em UML, da unidade de análise do PL/SQL Helper.

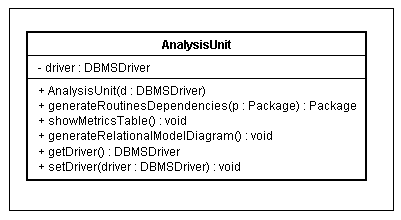


Figura 3.3 - Definição da entidade de análise do PL/SQL Helper

**3.3.2.1 Gráfico de Dependências**

A primeira funcionalidade selecionada para a seção de análise é de extrema utilidade para desenvolvedores que precisam analisar códigos legados: gráfico de dependências. Este gráfico mostra os relacionamentos entre as rotinas do sistema, informando visualmente quais rotinas são chamadas por outras, garantindo ao desenvolvedor, o conhecimento sobre as conseqüências das modificações. Uma vez que o programador tem consciência de quais rotinas podem sofrer alterações lógicas devido a alguma mudança externa, é possível minimizar os conflitos, isolando essas rotinas dependentes. A ferramenta PL/SQL Helper é capaz de exibir as dependências entre as rotinas de um pacote, entre as rotinas de vários pacotes e entre as rotinas definidas isoladamente. O usuário pode selecionar visualmente qual pacote ou rotina que deseja analisar. A ferramenta, então, exibe em tempo real, o código da rotina selecionada ou quais são as rotinas do pacote selecionado, permitindo modificações de código. Ao selecionar a opção para gerar o diagrama, a ferramenta processa os dados e abre uma janela exibindo o diagrama gerado. A imagem pode ser salva em formato JPG [38].

**3.3.2.2 Métricas**

Outra funcionalidade definida, para o módulo de análise, é a exibição de métricas. As métricas definidas para esta ferramenta também têm o objetivo de facilitar a manutenção de códigos legados. Número de linhas de código e número de linhas de comentários por rotina são campos analisados por esta seção, pois juntos podem exibir um comparativo que informa o quão bem documentado está o código do projeto. O número de linhas de código por rotina também pode mostrar o tamanho das rotinas mantidas no banco de dados e se é necessária uma análise para extrair sub-rotinas para uma melhor modularização. Também são exibidos o número total de pacotes presentes no banco, número total de rotinas (dividido entre procedimentos e funções) e o número de rotinas dependentes. Esses valores representam uma visão geral do entrelaçamento dos códigos PL/SQL. presentes na aplicação de BD.

**3.3.2.3 Diagrama Relacional**

A última funcionalidade selecionada para seção de análise é importante para a compreensão da estrutura de um banco de dados: o modelo relacional. Uma aplicação de banco de dados possui um modelo relacional, que representa a estrutura de tabelas do sistema e seus relacionamentos. Existem ferramentas como o Dia [51] e o JUDE [52] que criam e editam diagramas relacionais de maneira simples. Porém, a utilização deles faz com que seja necessária a atualização do diagrama sempre que houver uma mudança na estrutura de tabelas da aplicação de banco de dados. O PL/SQL Helper automatiza o processo de atualização do diagrama relacional, pois ele verifica em tempo real, o modelo de tabelas e exibe o diagrama para o usuário em forma de imagem. A imagem exibida pode ser salva em formato JPG.

**3.3.3 Refatoramento**

O módulo de refatoramento foi selecionado por ser uma atividade importante no processo de aperfeiçoamento de um sistema de banco de dados já codificado. As reestruturações podem levar à melhoria de desempenho, manutenção e atualização do sistema, bem como diminuir o tempo de adaptação de novos desenvolvedores. Para este módulo foram definidas duas funcionalidades: geração automática de fachada e renomeação de entidades. As definições destas funcionalidades podem ser visualizadas no modelo UML exibido na Figura 3.4.

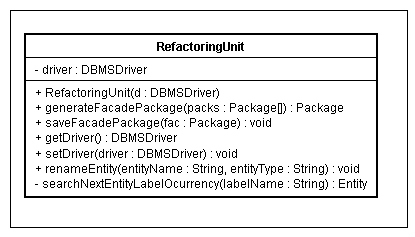


Figura 3.4 - Definição da entidade de refatoramento do PL/SQL Helper

**3.3.3.1 Fachada**

Projetos que possuem muitos pacotes especificados podem se tornar complexos demais se não possuírem uma modularização em camadas. As funcionalidades especificadas em cada pacote são acessadas explicitamente pela interface gráfica do sistema. Como visto na seção 2.3, a fachada resolve esse problema criando uma camada de abstração entre o módulo lógico e a interface do sistema. A ferramenta PL/SQL Helper incorpora um gerador de fachadas para os pacotes do sistema de bancos de dados. Esta funcionalidade consiste em procurar os diferentes pacotes de dados do projeto e gerar um pacote fachada automaticamente. Este pacote centraliza todas as funcionalidades implementadas nos demais pacotes, abstraindo sua localização no código de interface. O usuário pode visualizar o código do pacote gerado pela ferramenta e escolher a opção “Salvar” para gravá-lo no banco de dados.

**3.3.3.2 Renomeação**

Outro problema encontrado no desenvolvimento de aplicações de banco de dados é a definição dos rótulos. Com uma equipe grande de desenvolvedores, nomes não-padronizados para as entidades do sistema podem ser adotados. Tabelas, procedimentos, funções, variáveis e pacotes podem receber valores que não seguem um padrão comum. A ferramenta PL/SQL Helper provê a funcionalidade de renomear automaticamente rótulos de uma aplicação PL/SQL. Em um processo de reestruturação de código, pode ser necessário renomear entidades para rótulos definidos dentro de um padrão, e a funcionalidade de renomeação resolve os problemas que poderiam ser ocasionados numa alteração de rótulo, como visto na seção 2.4. O usuário seleciona visualmente qual entidade deseja renomear, define qual o novo valor do rótulo e seleciona a opção de “Renomear”. A ferramenta faz uma busca por todas as ocorrências do antigo valor no banco e altera a chamada para o novo valor.

**3.3.4 Testes**

Por fim, esta seção descreve o quarto módulo selecionado para a ferramenta: o módulo de testes. Ele foi escolhido por proporcionar uma proteção a falhas advindas de alterações no código. O módulo de testes possui duas funcionalidades principais: testes unitários [39,40] e testes de pacote [41]. Como visto na seção 2.5, a pesquisa feita sobre trabalhos correlatos que possuem um módulo de teste, resultou no estudo de um *framework* de código aberto para testes em códigos PL/SQL: o utPLSQL [2]. O PL/SQL Helper possui criação automática dos pacotes de testes necessários para realização dos testes unitários providos por este *framework*. O usuário somente define qual rotina ou pacote deve ser testado e a ferramenta gera o pacote para o teste com as funções necessárias. O usuário então preenche as funções com o código de verificação para os testes unitários, usando as asserções definidas pelo utPLSQL.

O esforço para incorporar o *framework* ao PL/SQL Helper consistiu na criação de um subsistema que automatizasse a instalação do utPLSQL no sistema de banco de dados do usuário. Além dessa funcionalidade, foi preciso que fossem implementados um módulo para automatização da criação dos pacotes de testes e um módulo de interface gráfica para utilização destas funcionalidades. A entidade de testes pode ser visualizada em modelo UML mostrado na Figura 3.5.

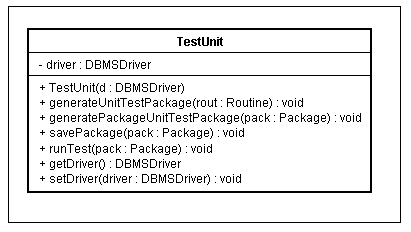


Figura 3.4 - Definição da entidade de teste do PL/SQL Helper

**3.4 Comparativo**

Esta seção exibe um comparativo entre os trabalhos correlatos, analisados na seção 2.5 sobre o estado da arte, e a ferramenta proposta por este trabalho. O quadro 3.1 exibe o resultado comparação feita, de acordo com as funcionalidades mostradas na seção 3.3.

É possível notar que as funcionalidades listadas no quadro 3.1 não estão presentes, reunidamente, nas diversas ferramentas analisadas. O ClearSQL, por exemplo, possui apenas a exibição de métricas, enquanto que o PLDOC contempla apenas as funcionalidades de documentação e o utPLSQL somente provê a realização de testes unitários. Por outro lado, o Toad implementa três funcionalidades de pós-codificação: métricas, testes unitários e diagrama relacional das tabelas. Apesar disso, só a ferramenta PL/SQL Helper provê criação de fachadas, renomeação de entidades e gráficos de dependências.

Quadro 3.1 – Comparativo de funcionalidades

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | ClearSQL | TOAD | PLDOC |  | utPLSQLPL/SQL Helper |
| Métricas | Sim | Sim | Não | Não | Sim |
| Testes Unitários | Não | Sim | Não | Sim | Sim |
| Diagrama Relacional de Tabelas | Não | Sim | Não | Não | Sim |
| Documentação de pacotes | Não | Não | Sim | Não | Sim |
| Documentação de Entidades Isoladas | Não | Não | Sim | Não | Sim |
| Gráfico de dependência das rotinas | Não | Não | Não | Não | Sim |
| Criação de Fachada | Não | Não | Não | Não | Sim |
| Renomeamento de entidades | Não | Não | Não | Não | Sim |
| Independência de Sistema Operacional | Não | Não | Sim | Sim | Sim |

**3.5 Conclusão**

Como vimos, a modularização em camadas tornou possível o desenvolvimento de uma ferramenta extensível. Além disso, com os estudos, foi possível definir um conjunto de funcionalidades para auxiliar no entendimento do código e na manutenção de códigos legados para projetos PL/SQL, até então não abordadas pelas ferramentas analisadas.

**4 Protótipo do PL/SQL Helper**

Este capítulo aborda o desenvolvimento do protótipo do sistema PL/SQL Helper que implementa um subconjunto das funcionalidades especificadas na arquitetura de software proposta no capítulo anterior. O protótipo foi implementado para validar algumas destas funcionalidades. Inicialmente, será dada uma introdução que explica como foi feita a escolha de cada funcionalidade implementada e em seguida, serão exibidos detalhes a respeito do funcionamento delas.

**4.1 Introdução**

Como especificado no capítulo anterior, foram definidos quatro módulos para a ferramenta: Documentação, Análise, Refatoramento e Teste. Dentro destes módulos, estão presentes dez funcionalidades diferentes que foram analisadas separadamente na seção 3.3. Foram escolhidas três destas dez funcionalidades para implementação no protótipo funcional: documentação de pacotes do módulo de documentação, gráfico de dependências do módulo de análise e geração de fachada do módulo de refatoramento. O gráfico de dependências e a geração automática de fachada, foram escolhidos para o módulo de análise e refatoramento, respectivamente, por não serem implementados por nenhuma das ferramentas analisadas no estudo sobre os trabalhos correlatos. A documentação de pacotes foi escolhida porque a idéia inicial era contemplar uma funcionalidade de cada módulo do PL/SQL Helper. Porém, para o módulo de testes, nenhuma funcionalidade foi escolhida para implementação. Para incorporação do utPLSQL ao PL/SQL Helper, era necessária a criação de um módulo instalador do *framework* no sistema do usuário, um módulo automatizador da criação de pacotes de testes e um módulo de interface gráfica para utilização das funcionalidades. Isto gerou uma necessidade de redução no escopo do desenvolvimento do protótipo de d, removendo de sua implementação uma funcionalidade do módulo de testes.

**4.2 Funcionalidades**

O protótipo funcional possui, além das três funcionalidades escolhidas, um sistema de conexão com o banco de dados. Portanto, ao todo, o protótipo possui quatro funções utilizadas pelo usuário do PL/SQL Helper.

**4.2.1 Inicialização**

Ao inicializar a aplicação, o usuário visualizará a tela principal do protótipo como podemos observar na figura 4.1.

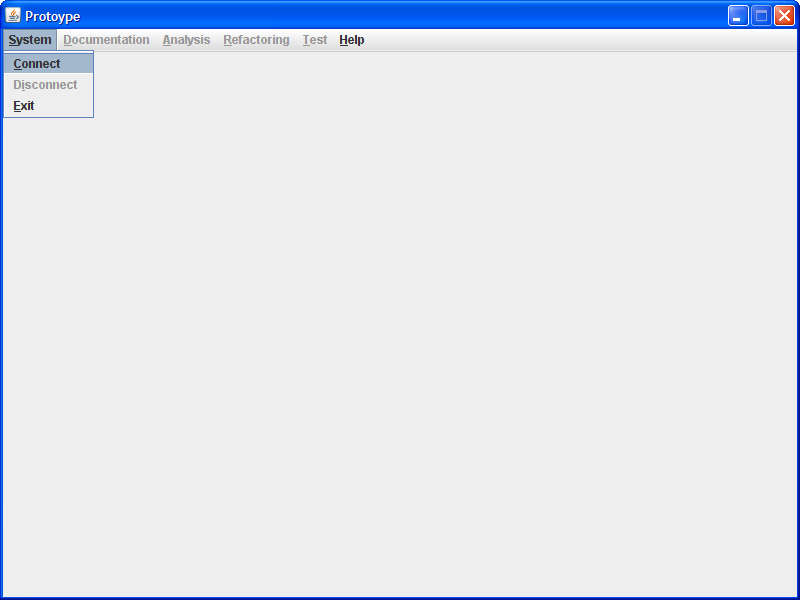
****

Figura 4.1 - Tela principal do protótipo

Na tela principal do sistema, mostrada na figura 4.1, é possível notar a presença da barra de tarefas. Esta barra é a via de acesso às funcionalidades da ferramenta PL/SQL Helper. A primeira opção da barra é o menu “*System*” que permite ao usuário, se conectar à base de dados pelo sub-menu “*Connect*”, se desconectar da base através do “*Disconnect*” e terminar a execução do programa pela opção “*Exit*”.

**4.2.2 Conexão**

Para acessar as funcionalidades do protótipo pertencentes aos menus “*Documentation*”, “*Analysis*” e “*Refactoring*”, é necessário estar devidamente conectado à base de dados. Acessando a opção “*Connect*” do menu “*System*”, a janela de conexão é aberta para o preenchimento dos dados do usuário. A ferramenta coleta o nome do usuário, a senha, a porta de conexão utilizada pelo sistema de BD, o nome do serviço e o IP no qual está localizado o servidor. A opção para selecionar o SGBD está habilitada, mas inicialmente só o valor ORACLE está disponível. A janela de conexão pode ser visualizada na figura 4.2.

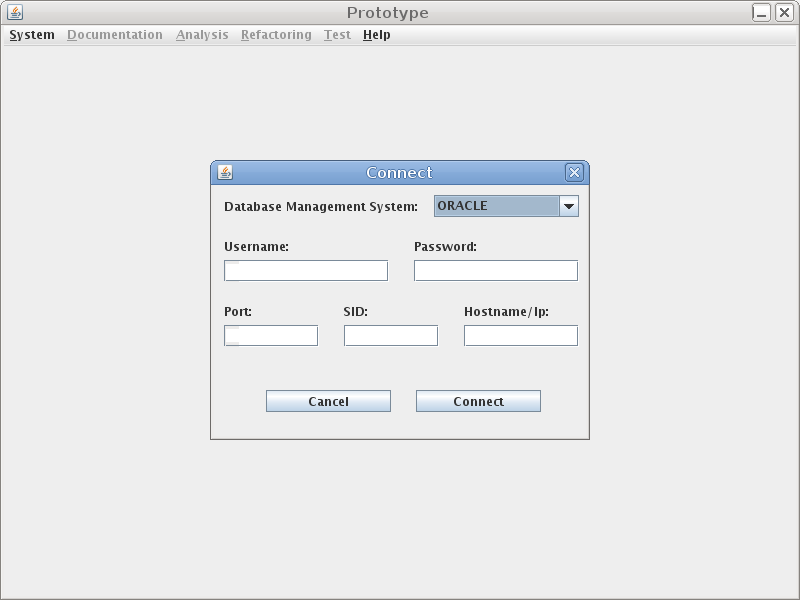


Figura 4.2 - Janela de conexão

**4.2.3 Documentação**

Uma vez conectado, o usuário pode utilizar qualquer uma das três funcionalidades implementadas para validação dos três módulos principais do sistema proposto: documentação, análise e refatoramento. A primeira delas que está disponível na barra de tarefas é a documentação de pacotes. Essa funcionalidade se localiza no menu “*Documentation*” com o nome de “*Generate Package Documentation*”. Uma vez selecionada a opção para documentação de pacotes, a ferramenta abrirá uma janela para as escolhas visuais do usuário, como é possível verificar na figura 4.3.

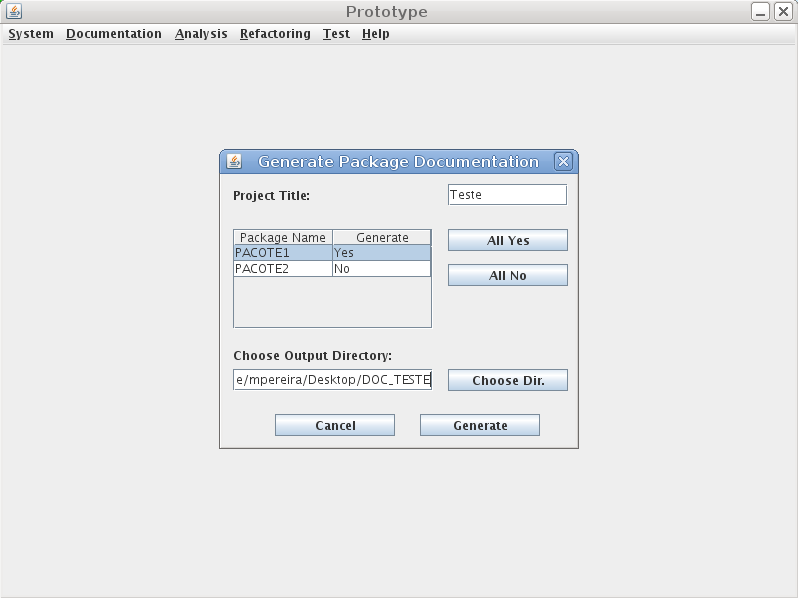


Figura 4.3 - Configurações de documentação

Pela janela exibida na Figura 4.3, o usuário especifica o título do projeto, para quais pacotes ele deseja gerar a documentação e onde serão armazenados os arquivos de documentação. Uma vez selecionados, o usuário clica na opção “*Generate*” e a ferramenta entra em processamento, informando o final da operação com uma mensagem na tela.

**4.2.4 Análise**

A segunda funcionalidade acessível pela barra de tarefas e exibida na tela principal do protótipo de sistema, que é mostrada na Figura 4.1, é a geração do gráfico de dependências. Localizada no menu “*Analysis*”, esta opção, ao ser selecionada, inicia um processamento que coleta todos os pacotes e rotinas do banco de dados. A execução desta tarefa acompanha uma barra de progresso para que o usuário consiga estimar o tempo de espera. Um exemplo da visualização desta barra é ilustrada pela Figura 4.4.

Ao finalizar o carregamento, a ferramenta irá exibir uma janela para que o usuário especifique para qual pacote, deve ser gerado o gráfico de dependências. Ao selecionar o pacote desejado, a ferramenta automaticamente exibe todas as rotinas contidas naquele pacote e seus respectivos códigos-fonte, como é possível visualizar na Figura 4.5.

Pode-se notar na Figura 4.5, que as opções para edição do código-fonte estão desabilitadas para esta versão para fins de redução do escopo do traabalho. Ao usuário selecionar a opção “*Selected Package*”, o sistema exibe o gráfico de dependência entre as rotinas do pacote especificado. Ao finalizar, a ferramenta exibirá uma janela com o gráfico de dependências gerado, como pode ser visto na Figura 4.6.

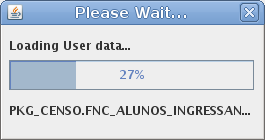


Figura 4.4 - Barra de progresso da geração do gráfico de dependências

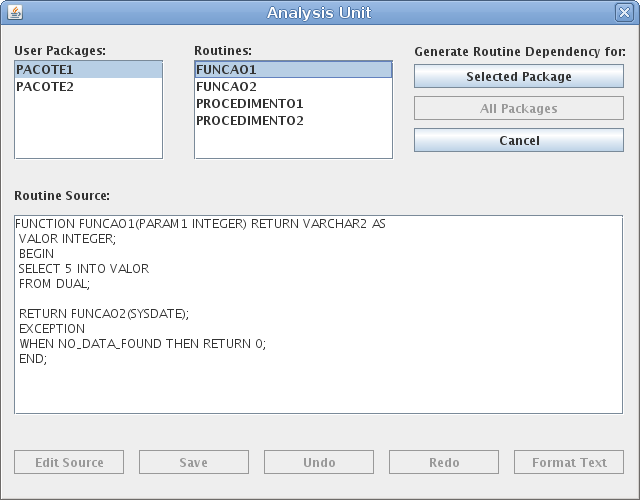


Figura 4.5 - Janela de configuração do gráfico de dependências

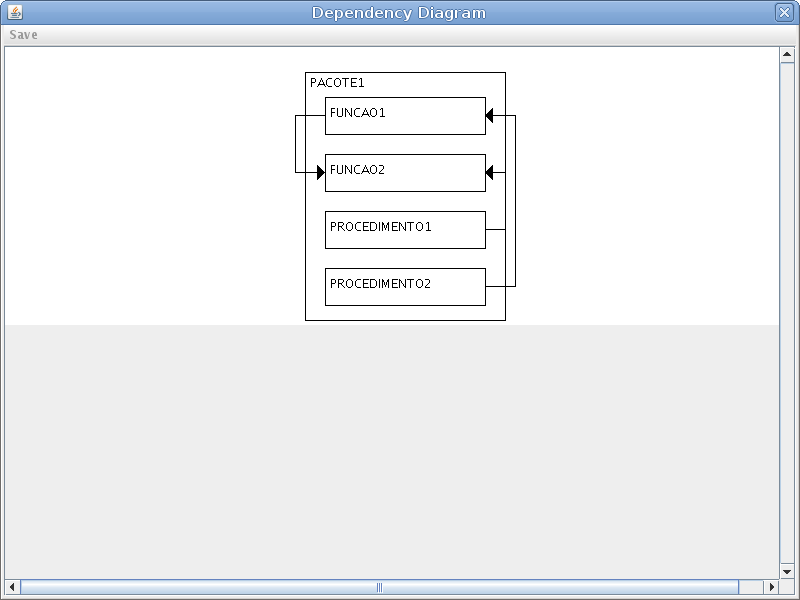


Figura 4.6 - Exemplo de diagrama de dependência gerado pela ferramenta PL/SQL Helper

**4.2.5 Refatoramento**

Outra funcionalidade disponível no protótipo do sistema PL/SQL Helper é a geração automática de fachada. Esta opção está disponível no menu “*Refactoring*” com o nome de “*Facade Creation*”. Ao ser selecionada esta funcionalidade, a ferramenta exibirá uma barra de progresso como a da Figura 4.4, indicando a realização de uma operação de carga de dados. Ao final desta operação, uma janela será exibida informando quais os pacotes estão definidos no esquema do banco de dados do usuário. O usuário pode selecionar a opção “*Create Facade*” que inicializa a atividade deste módulo. Após o processamento, o código gerado da fachada é exibido no campo de texto como pode ser visualizado na figura 9.0. Com o código gerado, o usuário pode selecionar a opção “*Save Facade*” para gravar o código do pacote no banco de dados.

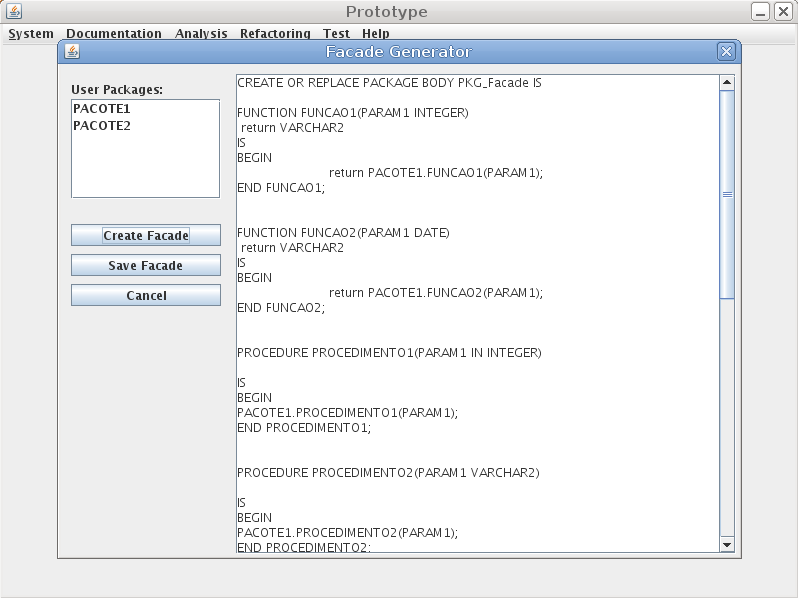


Figura 4.7 - Janela de refatoramento com o código da Fachada gerado

**4.3 Conclusão**

O protótipo foi desenvolvido para validação das principais idéias propostas no capítulo anterior e permitir a realização de um estudo de caso que é descrito no próximo capítulo. No projeto deste estudo de caso, três funcionalidades foram escolhidas: documentação de pacotes, gráfico de dependências de rotinas de um pacote, e geração automática de um pacote fachada, que foram definidos na seção 4.1. Todas estas funcionalidades estão acessíveis através de menus, botões e janelas de forma visual e rápida.

**5 Estudo de caso**

Este capítulo abordará a validação da ferramenta proposta por este trabalho. A validação será feita com base nos resultados obtidos com a utilização do sistema em um ambiente real de bancos de dados. O estudo foi feito junto ao Projeto SIG@ com o objetivo de resolver problemas de adaptação de novos integrantes ao projeto e de manutenção de código legado. Este capítulo está organizado como segue. Na seção 5.1, será apresentado o Projeto SIG@, alvo do estudo contido neste capítulo. Em seguida, na seção 5.2 será exibido um breve resumo a respeito do esforço necessário para o uso da ferramenta PL/SQL Helper no ambiente do nosso estudo de caso. Então, serão mostrados os resultados obtidos com o desenvolvimento desta aplicação. E por fim, na seção 5.4, será dada uma conclusão sobre os resultados do uso do PL/SQL Helper.

**5.1 Projeto SIG@**

O Sistema de Informações e Gestão Acadêmica (SIG@) é um *software* desenvolvido pelo Núcleo de Tecnologia da Informação (NTI) da Universidade Federal de Pernambuco (UFPE). Este *software* é o responsável pelas matrículas semestrais dos alunos da universidade, bem como votações, censo escolar e resolução de processos internos da instituição. A interface gráfica de utilização foi desenvolvida em JSP e acessa um banco de dados Oracle 10g. O sistema possui mais de quinhentas e cinquenta tabelas, cinquenta procedimentos isolados, vinte e quatro funções isoladas, dois pacotes lógicos contendo juntos mais de quinze rotinas dividas entre funções e procedimentos e cinquenta e oito usuários com os mais variados níveis de privilégio.

O grande número de componentes e dados dificulta a familiarização do desenvolvedor com a estrutura interna do sistema. Por este motivo, foi decidido fazer uso desta aplicação para validação do protótipo de sistema, detalhado no capítulo anterior, e verificar se a partir do PL/SQL Helper, é possível facilitar a adaptação de novos desenvolvedores e manutenção do projeto SIG@.

**5.2 Implantação**

A ferramenta foi implantada em um usuário especificamente criado para este trabalho, visando não exibir dados de segurança do projeto. O administrador criou um usuário com o nome “SPRINT4\_UFPE” para uso simulado da ferramenta. Como o sistema é portável e não precisa de instalação, a implantação é instantânea, desde que se tenha uma conta com senha para acessar o banco de dados. Utilizando um computador interno da rede usada pelos desenvolvedores do projeto, a ferramenta foi executada. Os computadores de desenvolvimento utilizam o sistema operacional OpenSuse versão 11.0 e acessam o banco de dados diretamente através da ferramenta Oracle SQL Developer [29].

**5.3 Resultados**

A documentação gerada produziu documentos para dois módulos do projeto SIG@: o SIG@ Censo e o SIG@ Matrícula. Os arquivos HTML produzidos automaticamente podem ser vistos nas Figuras 5.1, 5.2 e 5.3.

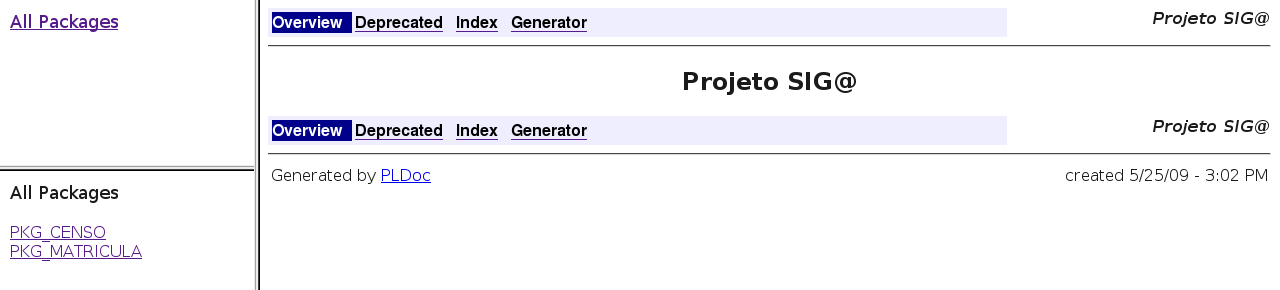


Figura 5.1 - Pagina Inicial da documentação Web do SIG@

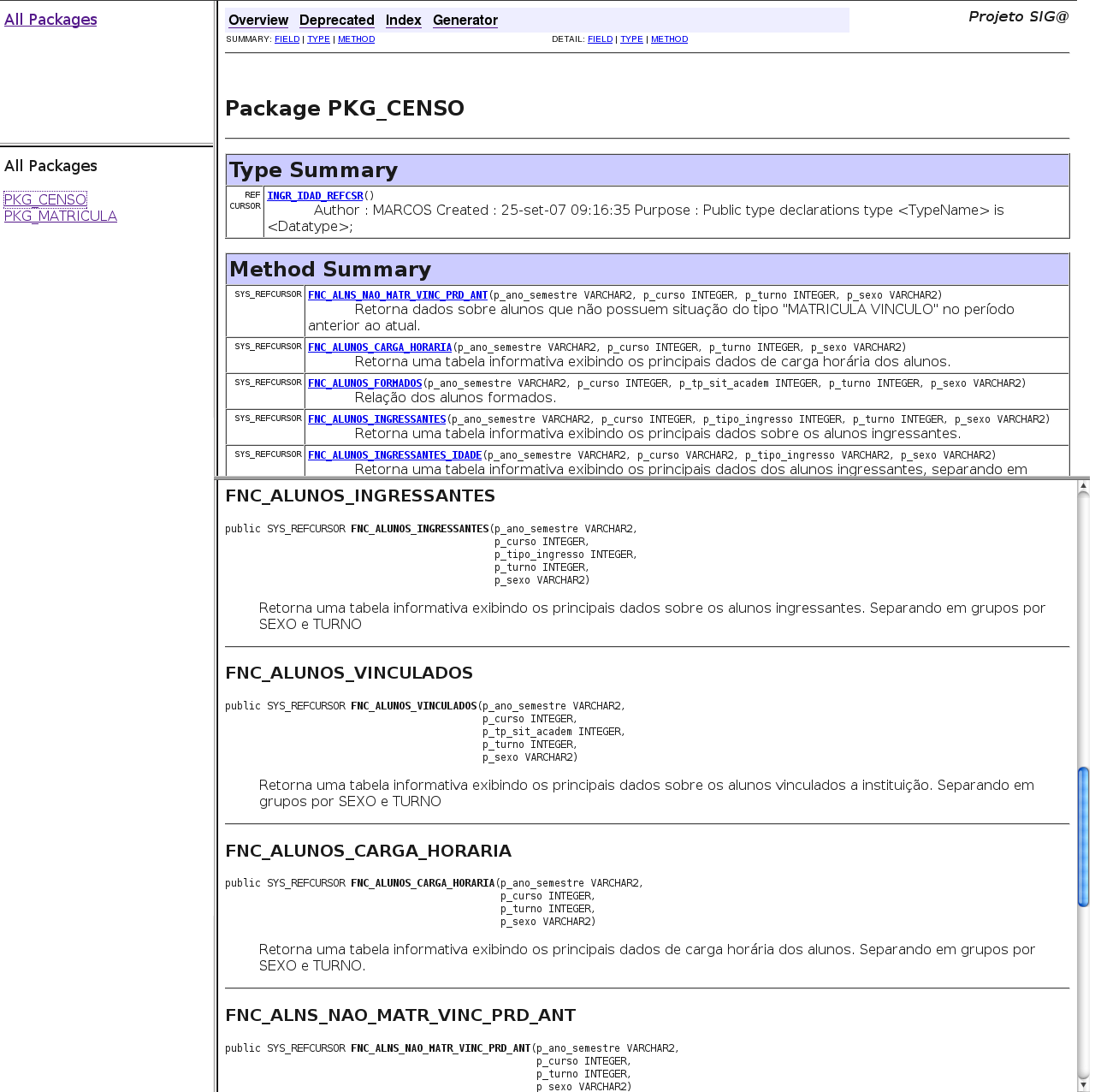


Figura 5.2 - Página de documentação para o Pacote de CENSO do SIG@

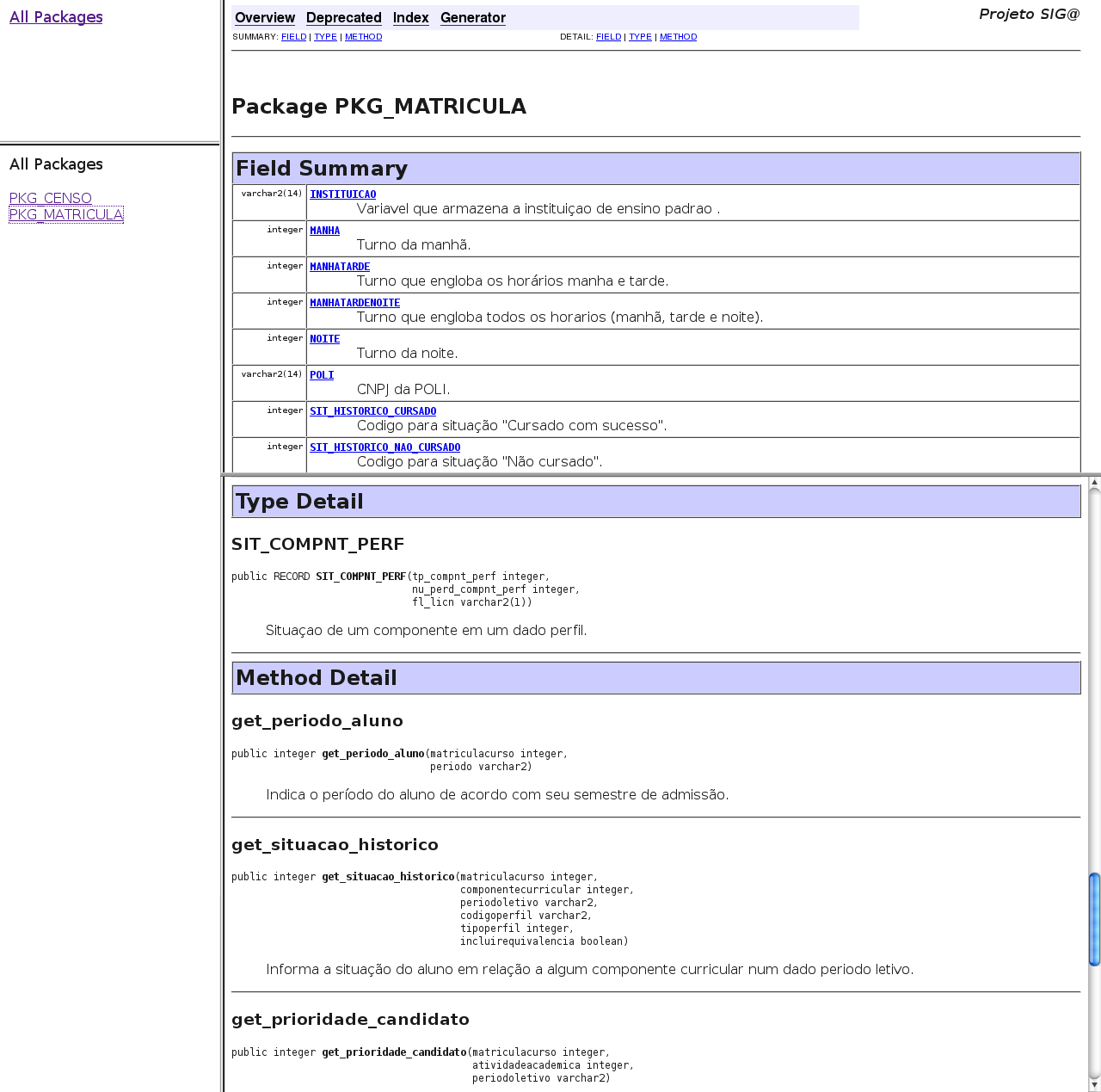


Figura 5.3 - Página de documentação para o Pacote de Matrícula do SIG@

É possível notar nas Figuras 5.1, 5.2 e 5.3, a existência dos *links* de navegação *web* para os diversos componentes presentes na documentação. Também é possível encontrar facilmente as definições das rotinas e atributos que compõem os pacotes do banco de dados, não necessitando para isto, a conexão direta de desenvolvedores com o BD.

Para a aplicação do módulo de análise, foram gerados gráficos de dependências para os dois pacotes lógicos do projeto: PKG\_CENSO e PKG\_MATRICULA. Os resultados podem ser visualizados nas Figuras 5.4 e 5.5. A Figura 5.4 exibe o grande número de rotinas do pacote PKG\_CENSO que dependem da função “get\_desc\_bl\_perf”.

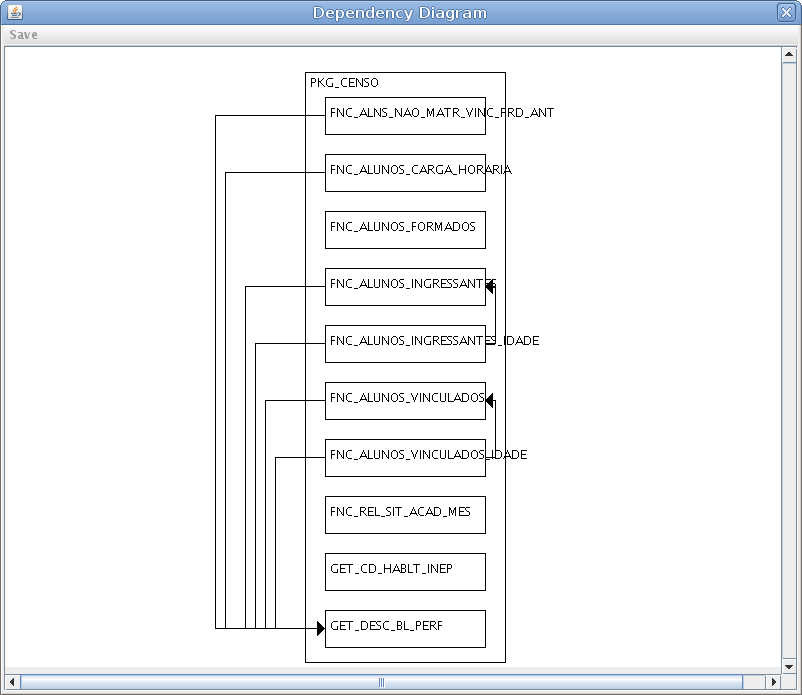


Figura 5.4 - Diagrama de dependências para o pacote CENSO do SIG@

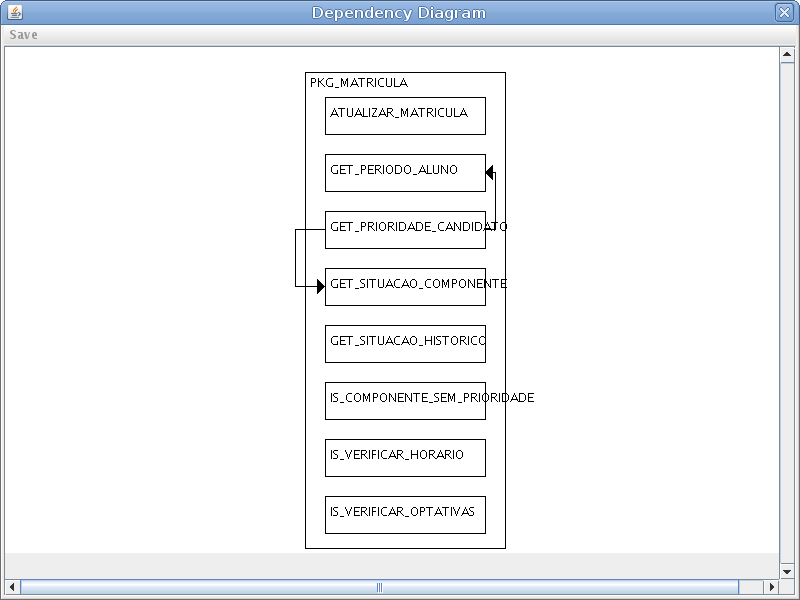


Figura 5.5 - Diagrama de dependências para o pacote Matrícula do SIG@

Para o módulo de refatoramento, a fachada gerada engloba as funcionalidades de ambos os módulos Censo e Matrícula. A fachada reuniu as funções e procedimentos destes pacotes num pacote chamado PKG\_FACHADA como pode ser visto na Figura 5.6.

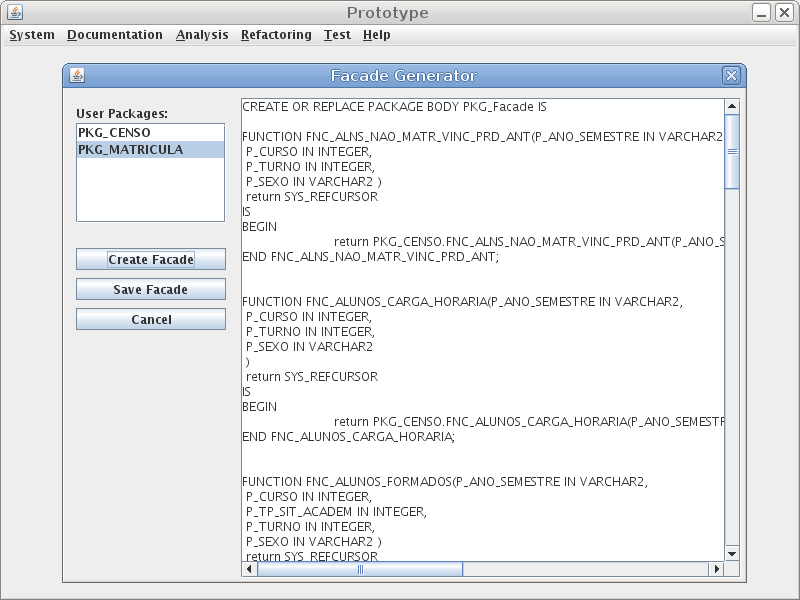


Figura 5.6 - Pacote Fachada gerado automaticamente pela ferramenta PL/SQL Helper

É possível notar na Figura 5.6, a exibição do corpo das funções do pacote fachada. Estes chamam a funcionalidade do pacote original, encontrando a localização da funcionalidade para as camadas superiores. A fachada foi criada diretamente no banco de dados como pode ser confirmado na Figura 5.7, a qual exibe a tela do Oracle SQL Developer utilizado pelos desenvolvedores (por motivos de segurança foram colocadas faixas vermelhas sobre os nomes dos outros usuários do projeto SIG@ que estão contidos nesta figura).

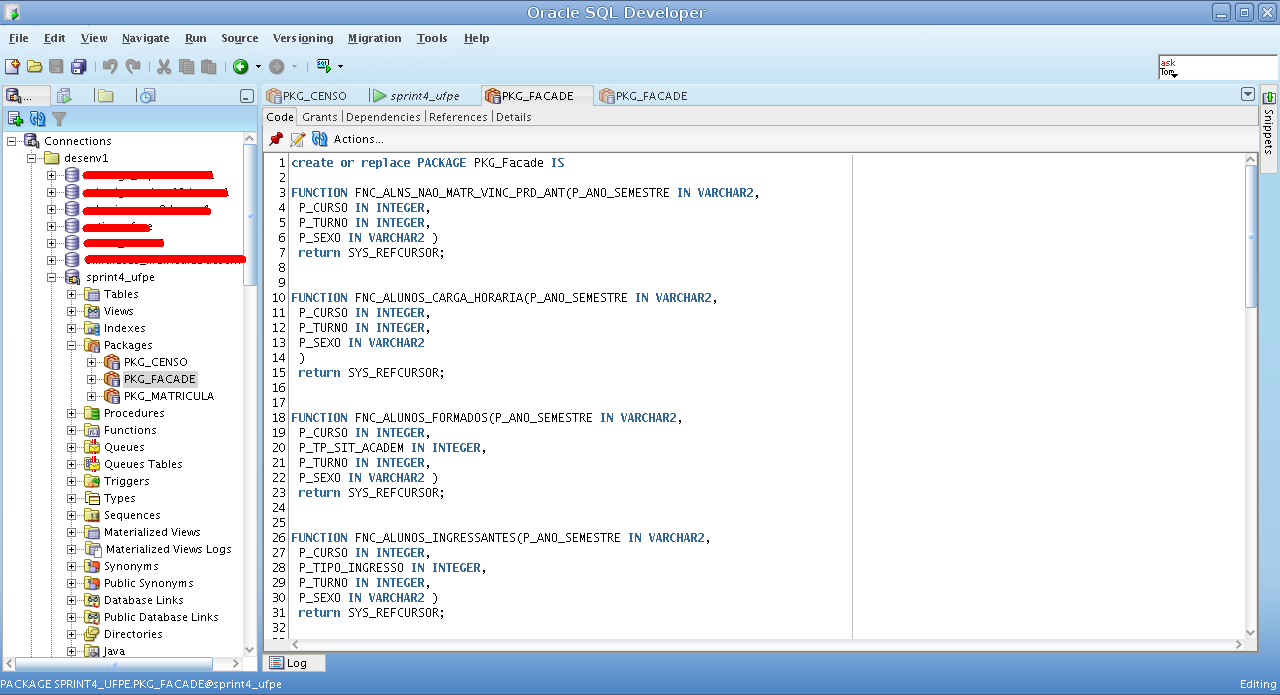


Figura 5.7 - SQL Developer exibindo o novo pacote lógico do usuário SPRINT4\_UFPE

**5.4 Conclusão**

Foi possível visualizar os comentários e a documentação das funções do banco de dados através de um navegador *web*, facilitando a aprendizagem dos desenvolvedores de código JSP novatos, uma vez que eles não precisam realizar várias buscas nos dados por meio de ferramentas de BD. Antes esses desenvolvedores necessitavam se conectar diretamente ao banco Oracle para compreender a tarefa realizada pela rotina ou o significado de um atributo. Também foi possível analisar a dependência das funções nos pacotes PKG\_CENSO e PKG\_MATRICULA, verificando a forte dependência das funções do pacote censo na função “get\_desc\_bl\_perf”. Por fim, foi possível garantir a transparência de localização das funções e procedimentos PL/SQL do banco Oracle para os desenvolvedores JSP. As funcionalidades agora podem ser chamadas de um pacote fachada que encapsula os diversos módulos do banco de dados, facilitando inclusive a incorporação de novos módulos.

**6 Conclusões**

Os objetivos do trabalho proposto neste documento consistem em: especificar e implementar uma arquitetura de ferramenta para a etapa de pós-codificação que ofereça funcionalidades de documentação, análise, refatoramento e testes ao desenvolvedor PL/SQL e implementar um protótipo para validação de um subconjunto destas funcionalidades. Com o estudo de caso e o protótipo desenvolvido, foi possível validar as funcionalidades do PL/SQL Helper junto a uma aplicação real: o Projeto SIG@. A documentação foi gerada corretamente, assim como os gráficos de dependências foram criados e a fachada foi compilada. Para o protótipo, não foi possível implementar todas as funcionalidades definidas no projeto da ferramenta, devido a uma necessidade de redução do escopo do trabalho, como visto na seção 4.1. Outro fator que impossibilitou a implementação de outras funcionalidades foi o fato de que funções e procedimentos, definidos dentro de pacotes, não estão presentes no dicionário de dados do Oracle. Isso significa que o SGBD Oracle trata as rotinas de um pacote como parte do pacote, ou seja, como texto. Não se tem conhecimento de ferramentas existentes no mercado e na academia que trabalhem com rotinas definidas dentro de pacotes do Oracle como o PL/SQL Helper. Foi necessário fazer extrações no código do pacote para recuperar isoladamente as funções e procedimentos por ele definidos, para que fosse possível verificar chamadas internas e gerar o diagrama de dependência e o pacote fachada.

**6.1 Contribuições**

Como principais contribuições para área de bancos de dados, é possível citar, primeiramente, a criação de uma interface gráfica para o programa PLDOC, que até então era invocado através de linhas de comando. O trabalho também contribuiu para o desenvolvimento de um módulo que exibe gráficos de dependências para rotinas definidas dentro de pacotes Oracle. Por fim, a geração automática de fachadas, modularizando o código da aplicação de banco de dados em camadas de acordo com padrão de projeto de software *Layer* [44], foi especificada e implementada para ser usada com o SGBD Oracle.

Além disso, este trabalho é original e consiste em uma primeira tentativa de especificar uma arquitetura de software que contemple funcionalidades para atividades pós-codificação como geração de documentação, gráfico de dependências, refatoramento e teste. As ferramentas existentes atualmente na academia e no mercado não possuem todas estas funcionalidades. A elicitação destas funcionalidades também é considerada uma contribuição.

**6.2 Trabalhos Futuros**

Alguns trabalhos futuros são listados abaixo como possíveis melhorias que podem ser realizadas no trabalho aqui descrito.

1. Para o módulo de documentação:
   1. Documentação de rotinas isoladas
   2. Documentação completa
2. Para o módulo de análise:
   1. Exibição do modelo relacional das tabelas em tempo de execução
   2. Visualização de métricas de análise
   3. Geração do gráfico de dependências entre rotinas de mais de um pacote
3. Para o módulo de refatoramento:
   1. Renomeação de entidades do banco de dados
4. Para o módulo de teste:
   1. Testes unitários para rotinas
   2. Testes para pacotes
5. Como funcionalidades diversas:
   1. A incorporação de edição de código-fonte em tempo real na janela de análise
   2. O armazenamento persistente das imagens geradas pelo gráfico de dependências
   3. Criação de novos *drivers* específicos para prover suporte a outros SGBD como MySQL e SQLServer.

**6.3 Referências Bibliográficas**

**[1]** <http://twiki.cin.ufpe.br/twiki/bin/view/TAES/LPS200802>

[2] utPLSQL: um framework de código aberto para testes PL/SQL http://utplsql.sourceforge.net/

**[3]** <http://oracle.su/owb.111/b31278/case_reuse_plsql_code.htm>

**[4]** <http://tylermuth.wordpress.com/2008/06/01/reusable-plsql/>

**[5]** Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J. **Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos**. Bookman, 2000.

**[6]** Alexander, C.; Ishikawa, S.; Silverstein, M.; Iacobson, M.; Fiksdahl-King, I.; Angel, S. A Pattern Language. Oxford University Press, New York, 1977

**[7]** Henninger, S. **Developing Domain Knowledge Through the Reuse of Project Experiences.** In: Symposium on Software Reusability. Proceedings…Seattle, 1995.

**[8]** Neighbors, J. **Draco: a method for engineering reusable software systems**. IEEE Computer Society Press, 1991.

**[9]** Simos, M. **Organization Domain Modeling (ODM): formalizing the core domain modeling lifecycle**. In: Symposium on Software Reusability. Proceedings…Seattle, 1995.

[10] http://www.combichemistry.com/glossary\_m.html

[11]Codd,E.F (Junho de 1970): "[A Relational Model of Data for Large Shared Data Banks"](http://www.acm.org/classics/nov95/toc.html) (Disponível em: <http://www.acm.org/classics/nov95/toc.html>)

[12]Akeel I. Din (Maio de 1994): Structured Query Language (SQL): a Practical Introduction

[13] Oracle Notes: The Oracle PL/SQL History (Disponível em: <http://www.oraclenotes.com/dba/list_plsql_history.cfm>)

[14]Observer Pattern Implementation in PL/SQL (Disponível em: <http://technology.amis.nl/blog/1123/design-patterns-in-plsql-implementing-the-observer-pattern>)

[15]Template Pattern Implementation in PL/SQL (Disponível em: http://blog.serpland.com/oracle/template-method-pattern-implemented-in-plsql)

[16] Dependency Injection Implementation in PL/SQL (Disponível em: <http://technology.amis.nl/blog/1086/dependency-injection-in-plsql-remove-hard-coded-dependencies-from-your-code>)

[17] PL/SQL Design Patterns (Disponível em: <http://blog.serpland.com/oracle/design-patterns-in-plsql-oracle>)

[18] Foy, Brian **The Facade Design Pattern** (Disponível em: http://www.theperlreview.com/Articles/v0i4/facade.pdf)

[19] Fowler, M ***Refactoring: Improving the Design of Existing Code*** (Disponível em: http://www.refactoring.com/)

[20] Imagem: **Refatoramento Wikipedia** (Disponível em: http://pt.wikipedia.org/wiki/Refatora%C3%A7%C3%A3o)

[21] Caetano, C. (maio de 2006) **Teste de Software for Dummies** (Disponível em: http://www.linhadecodigo.com.br/Artigo.aspx?id=1026)

[22] Minasi, M **Get the Bugs Out** (Disponível em: http://www.windowsitpro.com/Windows/Article/ArticleID/24255/24255.html)

[23] **Researchers Find Fewer Bugs In Linux** (Disponível em:  
http://www.developerpipeline.com/55800125)

[24] Pan, J **Software Testing**. Carnegie Mellon University, 1999 (Disponível em: http://www.ece.cmu.edu/~koopman/des\_s99/sw\_testing/)

[25] Hetzel, William C., ***The Complete Guide to Software Testing****, 2nd ed.* Publication info: Wellesley, Mass: QED Information Sciences, 1988.

[26] Myers, Glenford J., ***The art of software testing.*** *2004*

[27] Urman, S. **Oracle 9i: Programação PL/SQL**. 2002.

[28] Sommerville, Ian **Software Documentation** (Disponível em: <http://www.literateprogramming.com/documentation.pdf>)

[29] Oracle Technology Network. Oracle SQL Developer®. (Disponível em: <http://www.oracle.com/technology/products/database/sql_developer/index.html>)

[30] Allround Automations. PL/SQL Developer® (Disponível em: <http://www.allroundautomations.com/plsqldev.html>)

[31] Conquest Software Solutions. ClearSQL for Oracle® (Disponível em: <http://www.clearsql.com/>)

[32] Quest Software. ToadSoft® (Disponível em: http://www.toadsoft.com/)

[33] Sun: Java Language (Disponível em: http://java.sun.com/)

[34] Sun: Java Doc API (Disponível em: http://java.sun.com/j2se/javadoc/)

[35] PLDOC, an open-source utility for generating HTML documentatio. (Disponível em: http://pldoc.sourceforge.net/)

[36] OraEdit: an editor designed specifically for Oracle (Disponível em: <http://www.dkgas.com/oraedit.htm>)

[37] NTI, Núcleo de Tecnologia da Informação, UFPE. Projeto SIG@: Sistema de Informações e Gestão Acadêmica.

[38] JPEG: Joint Photographic Experts Group. (Disponível em: http://www.jpeg.org/jpeg/index.html)

[39] utPLSQL: Testing a procedure (Disponível em: <http://utplsql.sourceforge.net/Doc/testproc.html>)

[40] utPLSQL: Testing a function (Disponível em: <http://utplsql.sourceforge.net/Doc/testfunc.html>)

[41] utPLSQL: Testing a package (Disponível em: http://utplsql.sourceforge.net/Doc/testapi.html)

[42] Java Database Connectivity. JDBC. (Disponível em: http://java.sun.com/javase/technologies/database/index.jsp)

[43] JDBC: Architecture Overview (Disponível em: [http://java.sun.com/products/jdbc/overview.html#2](http://java.sun.com/products/jdbc/overview.html" \l "2))

[44] Black, S. Design Pattern: Layers

(Disponível em: http://stevenblack.com/PTN-Layers.html)

[45] Cohn, M. Uma introdução ao SCRUM. (tradução: Brod, C) Mountain Goat Software (Disponível em: <http://www.mountaingoatsoftware.com/system/hidden_asset/file/52/PortugueseScrum.pdf>)

[46] Extreme Programming. (Disponível em: <http://www.extremeprogramming.org/>)

[47] IBM. Rational Unified Process: Best Practices for Software

Development Teams. 2001. (Disponível em: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251\_bestpractices\_TP026B.pdf)

[48] Salgado, A. Fonseca, F. Times, V. Modelo Hierárquico. (Disponível em: <http://www.di.ufpe.br/~if559/slides/hierarquico.pdf>)

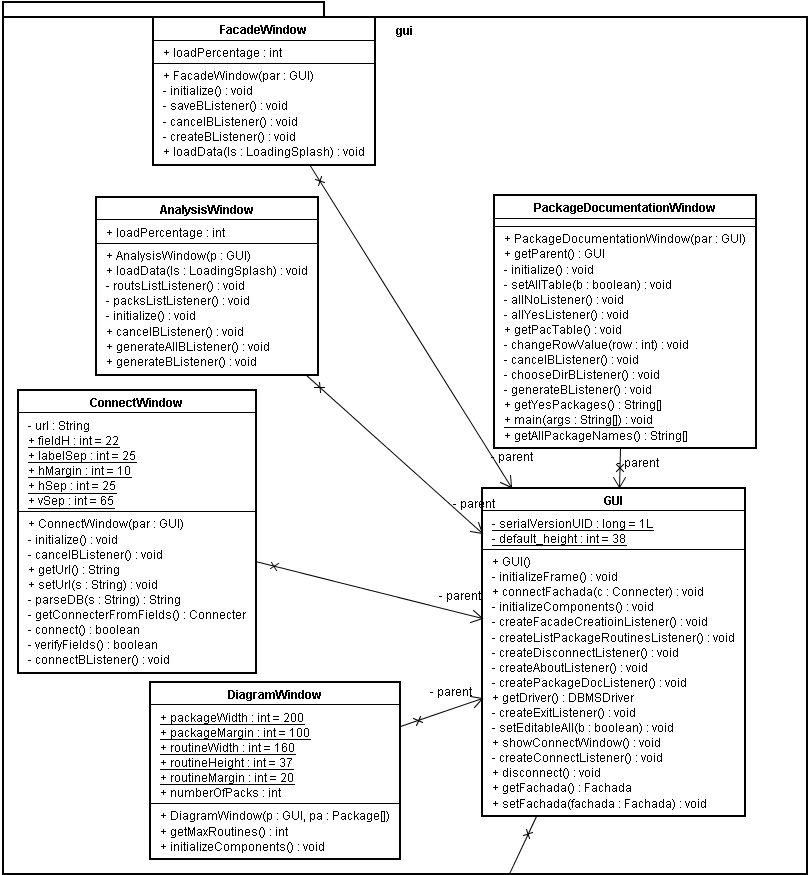
[49] Bellé, Adriane. Teste de Integração. Unicamp. (Disponível em: <http://www.dcc.unicamp.br/~eliane/Cursos/SeminarioTestes/testes_integracao.ppt>)

[50] Aguiar, D. Teste de Sistemas 2004. (Disponível em: http://www.dsc.ufcg.edu.br/~patricia/esii2004.2/aulas/teste\_sistemas.pdf)

[51]Dia, A GTK+ based diagram creation program (Disponível em: <http://dia-installer.de/index_en.html>, <http://live.gnome.org/Dia>)

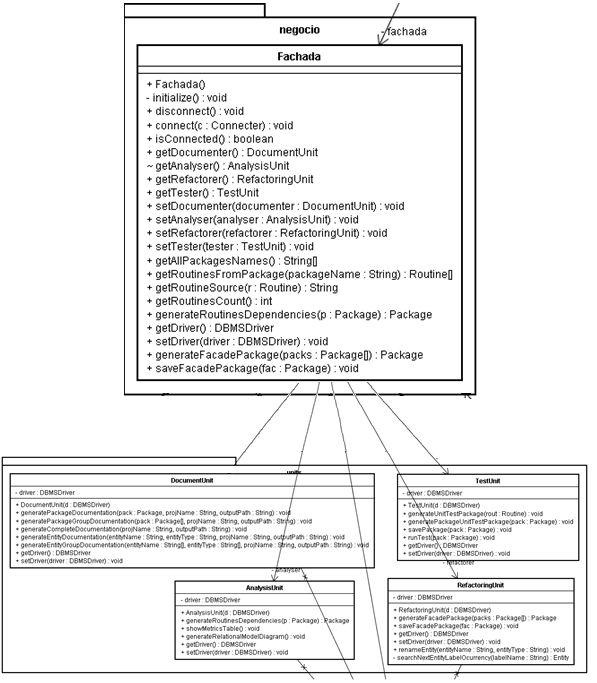
[52] JUDE, a system design tool. (Disponível em: <http://jude.change-vision.com/jude-web/index.html>)

**Anexo 1a**

****

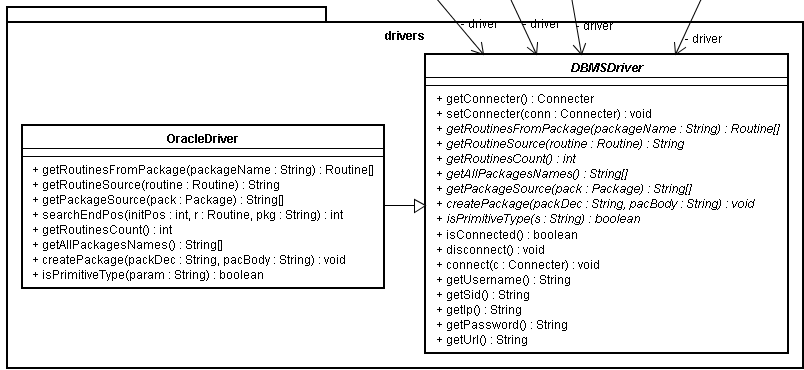
Modelo de classes UML do PL/SQL Helper (parte 1 – Camada de interfaces gráficas)

**Anexo 1b**

****

Modelo de classes UML do PL/SQL Helper (parte 2 – Camada de unidades lógicas)

**Anexo 1c**

****

Modelo de classes UML do PL/SQL Helper (parte 3 – Camada de acesso ao BD)

**Assinaturas**

|  |  |
| --- | --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
| Marcelo Victor Carneiro Pereira  (Aluno) | Valéria Cesário Times  (Orientador) |