UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

Ferramenta para Geração de Relatórios em Bancos de Dados utilizando Linguagens de Quarta Geração

ANDRÉ RICARDO ROLIM DOS REIS

Recife, Dezembro de 2011

ANDRÉ RICARDO ROLIM DOS REIS

Ferramenta para Geração de Relatórios em Bancos de Dados utilizando Linguagens de Quarta Geração

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do Grau de Bacharel em Ciência da Computação.

ORIENTADOR: FERNANDO DA FONSECA DE SOUZA

Recife, Dezembro de 2011

Aos meus pais, José e Zilsinéa.

AGRADECIMENTOS

Primeiramente agradeço a Deus por ter me proporcionado saúde e energia para a realização de todo o meu curso e deste trabalho.

Agradeço a meus pais, José e Zilsinéa, pelos valores a mim ensinados, pelo esforço incondicional para me proporcionar boa qualidade de vida e por serem o alicerce fundamental sem o qual eu não estaria aqui.

Agradeço ao meu irmão, Arthur Felipe, por estar sempre ao meu lado, meu companheiro de quarto de estudo e de dormir. Em breve ele estará concluindo seu curso também.

Agradeço a minha namorada Thaís, pelo apoio irrestrito em horas difíceis, minha companheira de todas as horas, por sempre ser tão carinhosa e incentivadora mesmo nos momentos mais complicados. O seu carinho e atenção significam muito para mim.

Agradeço ao meu orientador Fernando Fonseca que me deu total apoio e se mostrou totalmente prestativo para discutir o tema deste trabalho, mesmo eu tendo o procurado tardiamente.

Agradeço a meus amigos de escola (em especial a Felipe e Alan) que me acompanham até hoje num laço forte de amizade e camaradagem.

Agradeço a meus amigos do Centro de Informática, pelos bons e maus momentos passados juntos durante esta jornada, nas dificuldades das noites viradas e projetos concluídos (ou não) em cima da hora.

Agradeço também aos meus companheiros de estágio, pelo ambiente de trabalho agradável, pela troca de informações e conhecimentos, tão importantes para o meu desenvolvimento profissional.

Se você quer ser bem sucedido, precisa ter dedicação total,
buscar seu último limite e dar o melhor de si mesmo.
– AYRTON SENNA

RESUMO

As ferramentas existentes atualmente pertencentes à categoria *Reporting Tools* permitem que o usuário crie conjuntos de dados a partir de resultados de consultas SQL (*Structured Query Language*), mas não possuem opção para o usuário criar funções (*functions*) e procedimentos (*procedures*). Estas rotinas de linguagens de programação de quarta geração estendem o poder de SQL, adicionando construções comuns em linguagens procedurais,tornando-a bastante poderosa e tendo um papel importante na criação de relatórios mais complexos. Assim, o principal objetivo deste trabalho é estender o modelo atual de ferramentas de geração de relatórios para permitir o emprego de linguagens de quarta geração de sistemas de bancos de dados,sem a necessidade de escrita de código, a partir de elementos gráficos.

Palavras-chave: Banco de dados,*Reporting Tools*, Linguagens de programação de quarta geração.

ABSTRACT

Currently, tools belonging to theReporting Tools category allow their users to create data sets from the results of SQL (Structured Query Language) queries, but do not present option to create functions and procedures. These routines of fourth generation programming languages extend the SQL power, adding common constructions from procedural languages, making it sufficiently powerful to play an important role in the creation of more complex reports. Thus, the main objective of this work is to extend the current model of reports generation tools to allow the deployment of fourth generation languages of database systems from graphical elements, with no need for code writing.

Keywords: Database, Reporting Tools, Fourth-generation programming language.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

4GL	Fourth-Generation Language
API	Application programming interface
CASE	Computer-Aided Software Engineering
DSL	Domain-Specific Language
EGL	Epsilon Generation Language
EMF	Eclipse Modeling Framework
EOL	Epsilon Object Language
Epsilon	Extensible Platform of Integrated Languages for mOdelmaNagement
EVL	Epsilon Validation Language
EWL	Epsilon Wizard Language
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
HTML	HyperText Markup Language
PDF	Portable Document Format
PL/SQL	Procedural Language/Structured Query Language
SQL	Structured Query Language
UML	Unified Modeling Language
XML	eXtensible Markup Language

SUMÁRIO

1. INTRODUÇÃO

Um relatório é um trabalho textual que reúne um conjunto de informações com a intenção específica de reportar resultados parciais ou totais a respeito de determinados eventos de uma forma bem apresentável. Relatórios mais complexos podem incluir elementos adicionais, como gráficos, tabelas, figuras, imagens, resumos, sumários, referências, entre outros. Relatórios são usados no mundo dos negócios, governo, educação, ciência e em outros campos.

Com a expansão crescente da tecnologia da informação e o desejo de competitividade nas empresas, tem havido um aumento na utilização de recursos computacionais para produção de relatórios unificados que juntam diferentes visões da empresa em um só lugar. Este processo é denominado *Enterprise Reporting* e envolve a consulta de fontes de dados com diferentes modelos lógicos para produzir um relatório legível para humanos. Esses relatórios são atualizados regularmente e fornecem informações aos tomadores de decisão dentro de uma organização, por exemplo, auxiliando-os em seu trabalho[1].

1.1. Motivação

De acordo com a complexidade dos elementos que compõem o relatório, a sua criação pode ser uma tarefa complicada. Por isso, existem ferramentas para geração de relatórios (*Reporting Tools*) que permitem que o usuário crie relatórios de forma rápida e fácil. Essas ferramentas permitem que informações sejam carregadas de diferentes fontes de dados (bancos de dados e XML [2], por exemplo) e sejam apresentadas na forma de tabelas, listas, gráficos, textos, entre outros [3]. Caso a fonte de dados seja um banco de dados, informações podem ser extraídas através de consultas SQL [4] e rotinas de linguagens de programação de quarta geração, como procedimentos e funções. Após o seu desenvolvimento, o relatório formatado pode ser facilmente exportado para inúmeros formatos comuns de saída como PDF [5], *Excel*[6], *Word*[7] e HTML [8].

As linguagens de programação de quarta geração, ou 4GL[1] em sua abreviatura de origem inglesa, são linguagens com objetivos específicos, muitas vezes comparadas às linguagens de

---

[1] *Fourth-generation languages.*

domínio específico (DSLs[2]) [9], [10]. Algumas 4GL são específicas para banco de dados e são chamadas de *Database Query Languages*. Estas são linguagens bastante poderosas que permitem o acesso e manipulação de informações provenientes de uma base de dados e podem ser aplicadas na geração de relatórios, por exemplo. Um exemplo de linguagem de quarta geração específica para banco de dados é o Oracle PL/SQL que estende a linguagem SQL pela adição de construções encontradas em linguagens procedurais [11].

1.2. Objetivos

Este trabalho foi desenvolvido com o objetivo geral de criar um mecanismo para a criação de procedimentos a partir de elementos gráficos, sem a necessidade de escrita de código.Estes procedimentos criados serão aplicados na geração de relatórios, juntamente com alguma ferramenta de criação de relatórios (*Reporting Tool*). Sendo assim, a ferramenta gerada neste trabalho funcionará como uma extensão para qualquer outra ferramenta de *Reporting Tools* que dê suporteà utilização de procedimentos armazenados(*stored procedures*).

Este trabalho tem como objetivos específicos: 1) definir a melhor alternativa tecnológica disponível atualmente para o desenvolvimento de ferramentas gráficas de modelagem; 2) desenvolver um protótipo da ferramenta proposta que permita a modelagem gráfica do procedimento e geração de código; 3) visualizar o retorno do procedimento em forma de relatório utilizando alguma ferramenta de geração de relatórios.

1.3. Estrutura do trabalho

No presente capítulo, foram apresentadas a motivação e os objetivos deste trabalho de graduação. Nesta seção, são apresentados os demais capítulos.

O capítulo 2, ESTADO DA ARTE, apresenta uma análise de alguns sistemas de desenvolvimento de banco de dados que apresentam algum suporte à criação de procedimentos.

O capítulo 3, CONCEITOS ETECNOLOGIAS, apresenta a metodologia utilizada para o desenvolvimento do trabalho e a fundamentação teóricadas tecnologias necessárias na construção da ferramenta.

---

[2]*Domain-specific languages.*

O capítulo 4, A FERRAMENTA, apresenta o protótipo desenvolvido da ferramenta proposta para a criação de procedimentos utilizando elementos gráficos.

O capítulo 5, ESTUDO DE CASO, apresenta a utilização do protótipo desenvolvido em conjunto com uma ferramenta de geração de relatórios

No capítulo 6,CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS, são apresentadas as conclusões e contribuições deste trabalho, as dificuldades verificadas durante a sua produção e sugeridas possibilidades de trabalhos futuros.

Logo em seguida, as referências bibliográficas utilizadas na realização deste trabalho são listadas.

No Apêndice A encontra-se um tutorial de como criar um diagrama com a ferramenta desenvolvida neste trabalho.

2. ESTADO DA ARTE

Neste capítulo será analisado o suporte provido por dois sistemas de gerenciamento e desenvolvimento de banco de dados para a criação de procedimentos. Foram escolhidos o *Oracle SQL Developer* e o *MySQL Workbench* pela principal motivação de serem ferramentas gratuitas e amplamente utilizadas no desenvolvimento de aplicações que utilizam o Oracle ou o MySQL como SGBD.

2.1. Oracle SQL Developer

OracleSQLDeveloper[12] é uma ferramenta gráfica gratuita para desenvolvimento de banco de dadosno sistema Oracle. Com o *SQLDeveloper*, é possível navegar através dos objetos de banco de dados (tabelas, tipos, procedimentos, funções, entre outros), executar comandos e scripts SQL, e editar e depurar rotinas PL/SQL. Além disso, o programa fornece um conjunto de relatórios e também permite a criação de relatórios customizados. O *SQL Developer* é compatível com a versão 10g do sistema de banco de dados Oracle ou uma mais recente, e executa no Windows, Linux e Mac OSX[13].A Figura 2.1 ilustra a tela principal do sistema.

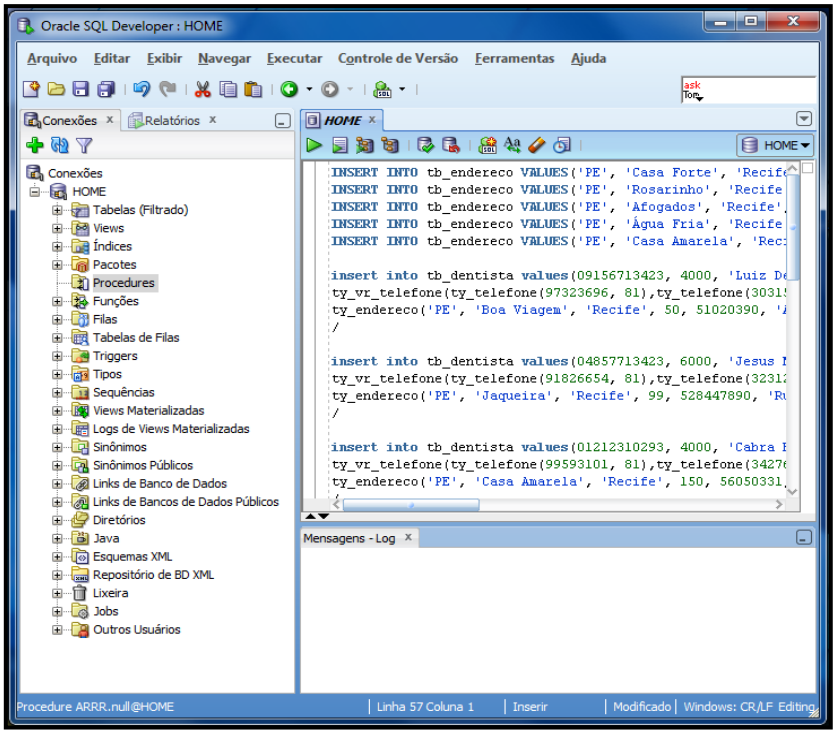

Figura 2.1 - Tela principal do Oracle SQL Developer.

Em relação ao suporte à criação de procedimentos PL/SQL, o *SQL Developer* permite que o desenvolvedor escolha um esquema no qual a rotina será inserida, defina seu nome e seus parâmetros. Para cada parâmetro é possível determinar o identificador (nome), o tipo de dados (VARCHAR2, NUMBER, DATE, CLOB ou BLOB), o tipo do argumento (IN, OUT, IN OUT) e o valor inicial padrão. A partir desses dados, o programa gera o código do procedimento que inclui o cabeçalho e o corpo vazio.Além disso, oreferido programapossui um editor de textos que destaca as construções de PL/SQL, indica erros e possui a função*autocomplete*[3], facilitando o desenvolvimento. A Figura 2.2 mostra a janela de criação de procedimentosdo sistema.

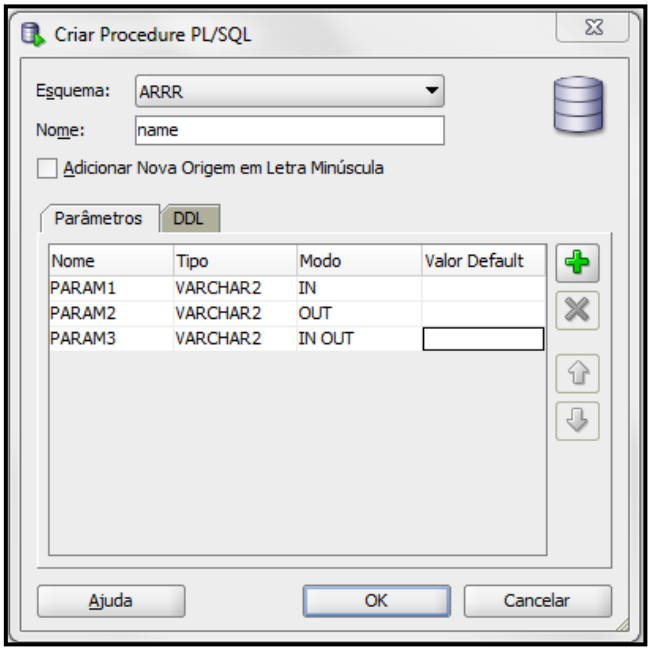

Figura 2.2 - Criação de procedimento com o Oracle SQL Developer.

[3]Função de preenchimento automático baseado em contexto.

2.2.MySQL Workbench

MySQL Workbench é uma ferramenta visual unificada para arquitetos, desenvolvedores e administradores de banco de dados. O programa fornece modelagem de dados, desenvolvimento de SQL, e ferramentas administrativas para configuração de servidor e administração de usuários, entre outros. Assim como o *OracleSQL Developer*, o *MySQL Workbench* está disponível para os sistemas operacionais Windows, Linux e Mac OS. O *MySQLWorkbench* é totalmente compatível com o sistema de banco de dados *MySQL*, a partir da versão 5.1[14]. A Figura 2.3 ilustra a tela principal do sistema.

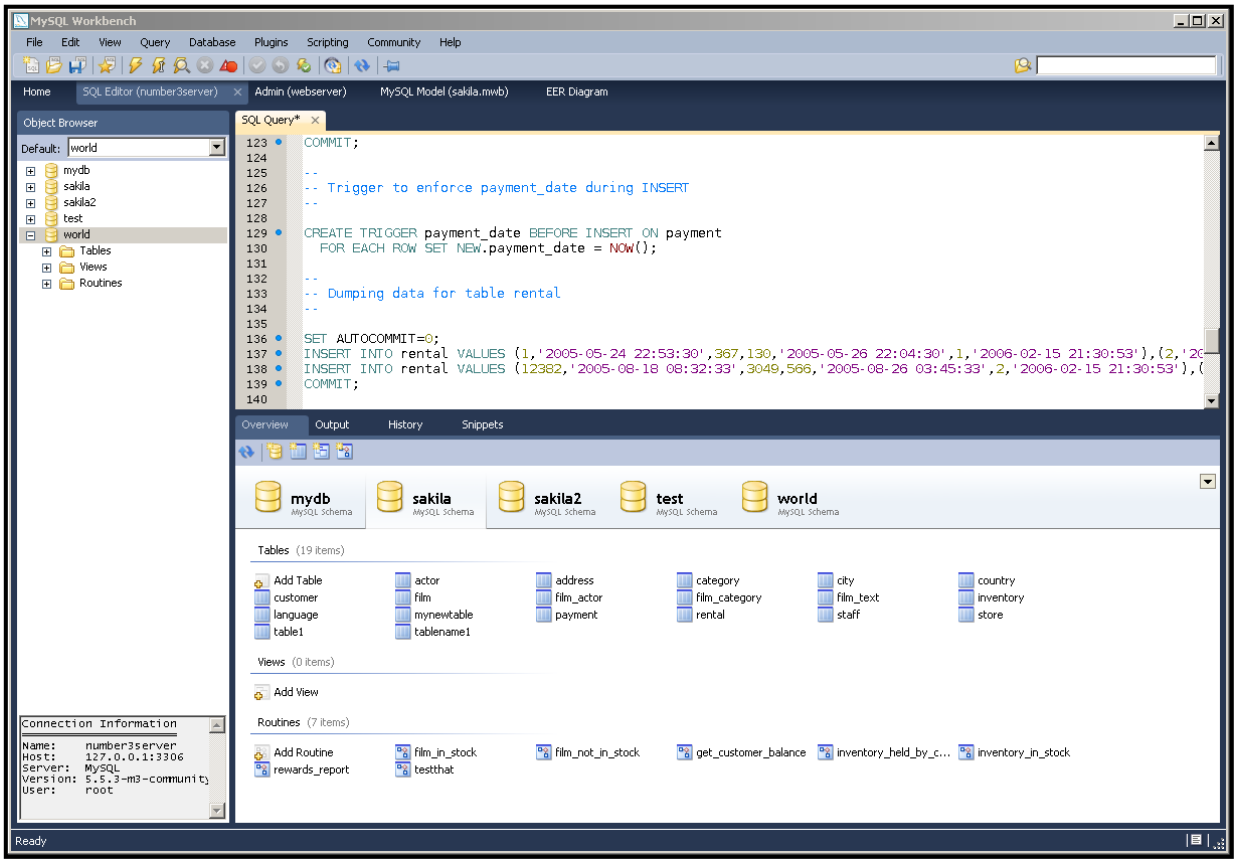

Figura 2.3 - Tela principal do MySQL Workbench.

O suporte à criação de procedimentos do *MySQL Workbench* é mais restrito que o encontrado no *SQL Developer*. Ao tentar criar um novo procedimento, o programa gera automaticamente a estrutura básica da rotina com cabeçalho e corpo vazio. Não há o mesmo suporte à definição do nome e parâmetros do procedimento, sendo necessário editar o código diretamente. O editor de textos do *MySQLWorkbench* também destaca as construções da

linguagem e indica quando o código contém erros, embora não possua função *autocomplete*. A Figura 2.4 mostra a criação de um procedimento utilizando o sistema.

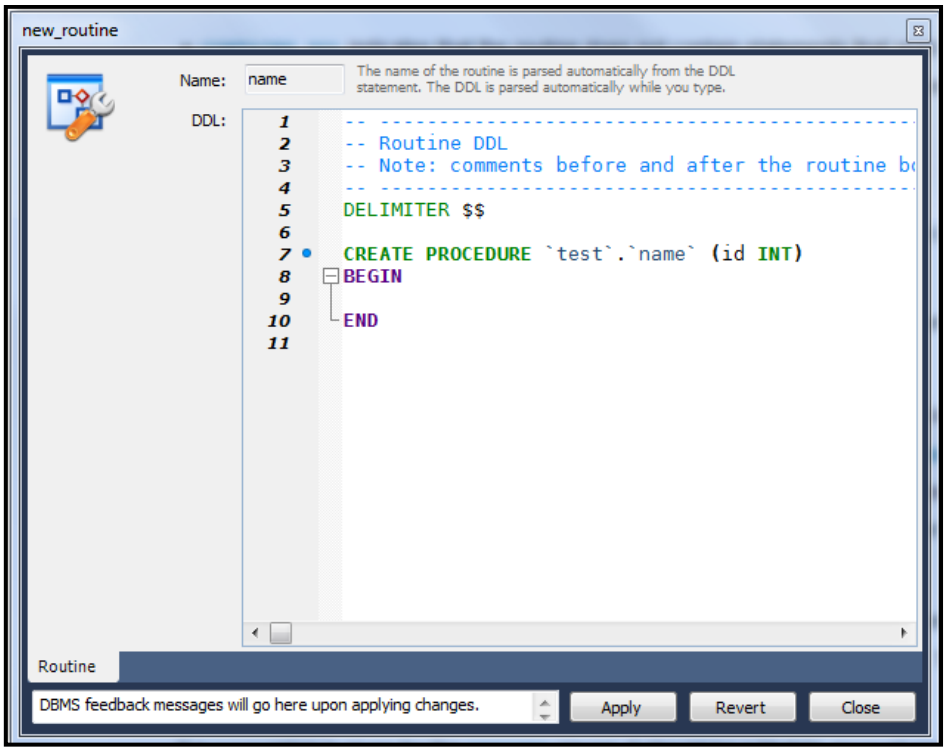

Figura 2.4 - Criação de procedimento com o MySQL Workbench.

2.3.Resumo das Características

Nesta seção será apresentado um quadro com as principais características dos sistemas analisados relativas ao suporte à criação de procedimentos. O Quadro 1 apresenta um resumo das característicasdo *Oracle SQL Developer* e do *MySQL Workbench*.

Quadro 1 - Resumo das principais características dos sistemas analisados.

Característica	Oracle SQL Developer	MySQL Workbench
Definição de parâmetros da rotina.	Possui	Não Possui
Definição de corpo da rotina.	Não Possui	Não Possui
Destaque para construções da linguagem.	Possui	Possui
Editor de texto com função *autocomplete*.	Possui	Não Possui
Editor de texto com indicação de erros.	Possui	Possui

É possível ver claramente que o *Oracle SQL Developer*comparado ao *MySQL Workbench*possui mais funcionalidades que auxiliam na criação de um procedimento, sem necessidade de edição direta de código. No entanto, nenhum dos dois programas analisados possui suporte à definição do corpo de um procedimento. Esta é a característica desejável que será levada em consideraçãodurante a criação da ferramenta proposta neste documento.

O próximo capítulo se dedica a apresentação dos conceitos, metodologia e tecnologia empregados no desenvolvimento do trabalho proposto.

3. CONCEITOS ETECNOLOGIAS

Para elaboração da proposta de uma ferramenta para modelagem de procedimentos com elementos gráficos, foi realizado primeiramente um estudo acerca das tecnologias necessárias para o seu desenvolvimento.Na seção 3.1 será apresentado o conjunto de tecnologias utilizado para desenvolvimento do protótipo e na seção 3.2 será apresentado um estudo acerca da linguagem Oracle PL/SQL, a linguagem escolhida para o desenvolvimento do protótipo.

3.1.Eclipse Modeling Project

Para o desenvolvimento da ferramenta proposta foi utilizado um conjunto unificado de *frameworks* de modelagem, ferramentas e implementações de padrões encontrados na comunidade Eclipse[15]. Dentre eles destacam-se os seguintes *frameworks*: EMF (*Eclipse Modeling Framework*)[16]e o GMF (GraphicalModeling Framework)[17] que se destinam ao desenvolvimento de metamodelos e editores gráficos para a criação de *ferramentas CASE*. Além dos frameworks citados, foi utilizada a plataforma Epsilon (*Extensible Platform ofIntegratedLanguages for mOdelmaNagement)*[18] que provê um conjunto de linguagens destinadas a tarefas de gerenciamento de metamodelos. Os próximos tópicos apresentarão uma visão geral dos recursos acima citados.

3.1.1. *Eclipse Modeling Framework* (EMF)

O EMF, como a própria sigla indica, é um framework de modelagem para a plataforma Eclipse. Ele utiliza as tecnologias Java[19] e XML para a geração de ferramentas e outras aplicações baseadas em modelo. O EMF facilita a tarefa de transformar modelos em códigos Java corretos, eficientes e facilmente customizáveis. Além da geração de código, o framework provê a capacidade de salvar objetos diretamente em XML, formato padrão para representação de dados[20].

O EMF é composto por três elementos fundamentais: 1) *EMF Core*, que inclui um metamodelo (*Ecore*) para descrição de modelos, além de suporte em tempo de execução para

estes, incluindo notificações de alteração, suporte a persistência e uma API[4] bastante eficiente para manipulação genérica dos objetos EMF; 2) *EMF.Edit*, que inclui classes genéricas reusáveis para a construção de editores para os modelos EMF; e 3) *EMF.Codegen*, que é responsável pela geração do código necessário para a criação dos modelos e editores gráficos da aplicação[20]. Os três elementos podem ser visualizados na Figura 3.1 da arquitetura EMF.

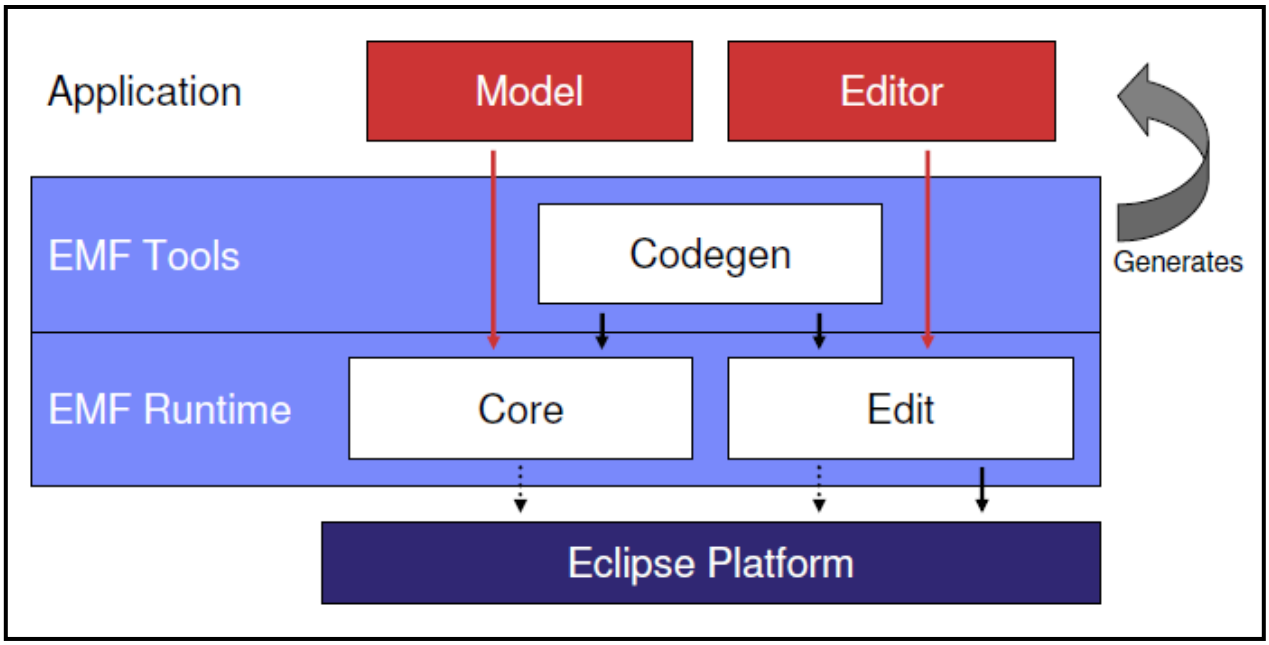

Figura 3.1 - Arquitetura EMF (EclipseCon 2008 - IBM Corp.)

O metamodelo*Ecore* é formado pelas seguintes principais componentes: *EClass*, utilizado para representar uma metaclasse; *EAttribute*, utilizado para representar um atributo de uma *EClass*; *EReference*, utilizado para representar associações entre classes; e *EEnum*, utilizado para representar enumerações[20].

O *framework* EMF dá suporte a três níveis de geração de código: Modelo, que provê classes e interfaces codificadas em Java para todos os elementos pertencentes ao metamodelo; Adaptadores, que geram as classes de implementação capazes de adaptar as metaclasses do modelo para edição e visualização; Editor, que produz uma estrutura básica apropriada do modelo que será utilizada na fase de geração do editor gráfico[20].

---

[4]Conjunto de instruções e padrões de programação para acesso a um aplicativo de software.

3.1.2. *GraphicalModeling Framework* (GMF)

O GMF permite o desenvolvimento de editores gráficos a partir de metamodelos definidos em EMF. O GMF também utiliza o *framework*GEF[21] que fornece recursos tecnológicos necessários para a concepção de ricos editores gráficos e visualizações para o Eclipse.

Para o desenvolvimento de editores gráficos usando GMF é necessário seguir um processo bem definido. Este processo é executado com a ajuda de um painel chamado *GMF Dashboard* que serve como um guia para o projetista[22][23]. O *GMFDashboard* é ilustrado na Figura3.2. É possível observar que a geração de um editor gráfico GMF contém seis passos:

1. ***Domain model*** -representa o metamodelo utilizado para criar o editor gráfico. É possível importá-lo de diversos tipos de fontes: código Java com anotações, modelo *Ecore*, modelo de classe do Rational Rose[24], modelo UML[25] ou esquema XML. Neste projeto foi utilizado o metamodelo*Ecore* do EMF;
2. ***Domain GenModel*** -arquivo com terminação.*genmodel* usado para gerar o código do *domainmodel* com EMF;
3. ***GraphicalDefModel*** - arquivo com terminação .*gmfgrafh* usado para definir os elementos gráficos que representarão cada um dos objetos do *domainmodel*;
4. ***ToolingDefModel*** -arquivo com terminação .*gmftool* usado para definir a paleta de ferramentas usada no editor gráfico. As ferramentas pertencentes à paleta podem ser usadas para seleção, criação de objetos e qualquer outra ação a ser realizada no diagrama;
5. ***MappingModel*** -arquivo com terminação .*gmfmap* que conecta o *domainmodel*, o *graphicalmodel* (.*gmfgraph*) e o *toolingmodel*(.*gmftool*); e
6. ***Diagram Editor GenModel*** -arquivo final com terminação .*gmfgen* usado para gerar o editor gráfico GMF.

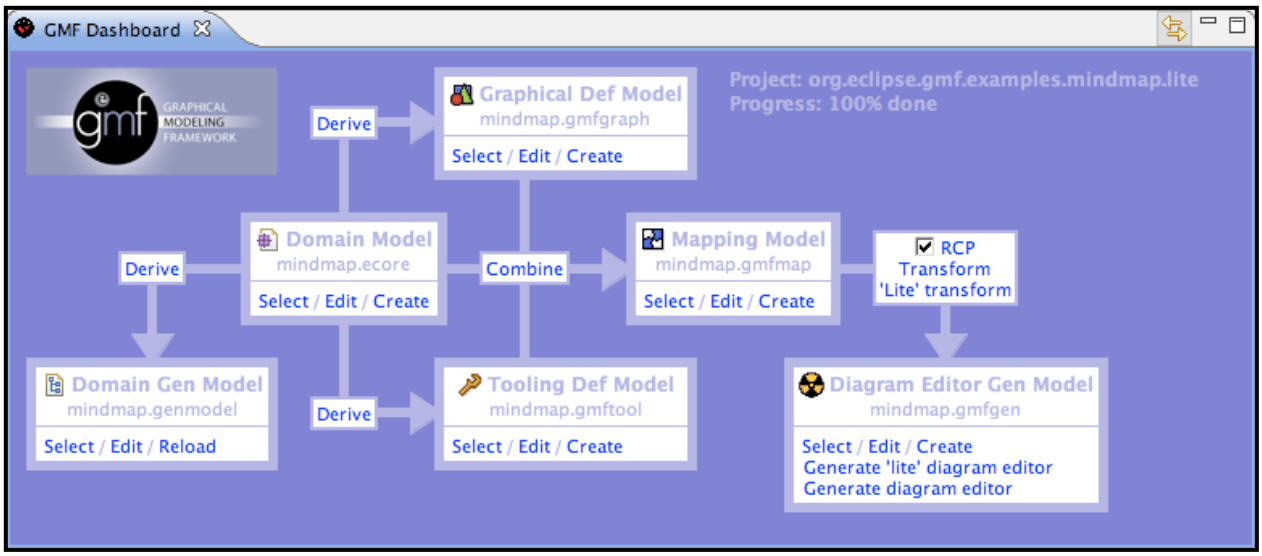

Figura3.2 - GMF Dashboard(Eclipse Foundation)

3.1.3. *Epsilon*

*Epsilon*engloba um conjunto de linguagens de programação destinadas a tarefas específicas que facilitam otrabalho de gerenciamento de modelos EMF. Algumas destas tarefas são: geração de código, transformação entre modelos, validação, comparação, migração, *merging* e *refactoring* de modelos[26]. Dentre as linguagens e ferramentas da família *Epsilon*, destacam-se *EuGENia*[27], *Emfatic*[28][29], EOL (*EpsilonObjectLanguage*)[30], EVL (*EpsilonValidationLanguage*)[31], EGL (*EpsilonGenerationLanguage*)[32] e EWL (*EpsilonWizardLanguage*) [33]que foram utilizadas durante o desenvolvimento deste trabalho.

EuGENia é uma ferramenta que facilita o trabalho de geração de editores gráficos em GMF, uma tarefa que normalmente seria complexa e árdua. *EuGENia* automatiza o processo de criação da ferramenta CASE a partir de um simples metamodelo*Ecore* com anotações, expresso usando a linguagem *Emfatic*[27].

Emfatic é uma linguagem de representação de modelos EMF *Ecore*, usando uma sintaxe textual compacta e legível semelhante à linguagem Java[29]. Ela dá suporte à definição de metaclasses, atributos de metaclasses, associações entre metaclasses e enumerações, além de outros elementos do modelo EMF *Ecore*. É possível gerar o modelo *Ecore* (arquivo *.ecore*) a partir de um arquivo *Emfatic* (representado pela extensão *.emf*) e vice-versa. Existe um mapeamento entre declarações *Emfatic* e construções *Ecore*, isto é, para cada declaração*Emfatic* há uma construção *Ecore* correspondente.

O metamodelo escrito em *Emfatic* pode ser enriquecido com anotações *EuGENia*que permitem definir como será a representação gráfica de cada metaclasse.Essas podem ser representadas como nós ou conexões (*links*) no diagrama resultante. Também é permitido definir qual figura representará cada nó (retângulo, elipse, entre outros) e como será desenhada cada conexão (extremidade com seta, quadrado ou losango, por exemplo), entre outras configurações. Todas estas definições são posteriormente mapeadas automaticamente em modelos GMF[27]. Todos os modelos encontrados no *Dashboard* GMF são gerados a partir desse metamodelo. O Anexo A apresenta um exemplo de metamodelo definido utilizando *Emfatic*.

EVL é uma linguagem que pode ser usada para, de maneira fácil e eficiente, adicionar validações e pequenos ajustes no editor gráfico GMF. Em outras palavras, é possível especificar e avaliar restrições no metamodelo utilizando EVL[26]. Um exemplo de código escrito em EVL pode ser visto na Figura 3.3. Em EVL, especificações de validação são organizadas em módulos:

- **Contexto (*context*) –** Um contexto especifica em quais tipos de instânciasas invariantes serão aplicadas. Na Figura 3.3, o contexto é encontrado na linha 1;
- **Invariante (*invariant*) –** Cada invariante EVL define um nome e uma verificação (*check*). Além disso, a invariante pode definir uma mensagem que deve ser mostrada ao usuário quando acontecer alguma falha de restrição. Existem duas subclasses de ,9, mensagens são encontradas nas linhas 4 e 10;
- **Guarda (*guard*) –** Elementos utilizados para restringir ainda mais o domínio de aplicabilidade das invariantes. Um guarda pode ser aplicado em termos de contexto, limitando a aplicabilidade de todas as invariantes do contexto, ou em termos de invariantes, limitando a aplicabilidade de uma invariante específica. Na Figura 3.3, o guarda é encontrado na linha 8;
- **Correção (*fix*) –** Uma correção basicamente define as ações que serão tomadas para reparar a inconsistência identificada. Na Figura 3.3, a correção é encontrada na linha 12;
- **Restrição (*constraint*) –** Restriçõessão usadas para capturar erros críticos que invalidam o modelo. Na Figura 3.3, a restrição é encontrada na linha 2;
- **Crítica (*critique*) –** Ao contrário das restrições, críticas são usadas para capturar condições não críticas do modelo, ou seja, condições que não o invalidam, mas devem ser apresentadas ao usuário. Na Figura 3.3, a crítica é encontrada na linha 7.

validation.evl

```
1  context File {
2    constraint HasName {
3        check : self.name.isDefined()
4        message : 'Unnamed ' + self.eClass().name + ' not allowed'
5    }
6
7    critique NameStartsWithCapital {
8        guard : self.satisfies('HasName')
9        check : self.name.firstToUpperCase() = self.name
10       message : 'Folder ' + self.name +
11           ' should start with an upper-case letter'
12       fix {
13           title : 'Rename to ' + self.name.firstToUpperCase()
14           do {
15               self.name := self.name.firstToUpperCase();
16           }
17       }
18   }
19 }
20
```

Figura 3.3 - Trecho de código em EVL.

EGLprovê uma linguagem baseada em *templates*[5]com várias funcionalidades que simplificam a tarefa de transformar modelos em textos (códigos). Um programa EGL consiste de uma ou mais seções estáticas ou dinâmicas. O conteúdo de seções estáticas é transformado na íntegra e aparecem diretamente no texto criado, enquanto o conteúdo de seções dinâmicas é executado e usado para controlar o texto que é criado. Uma seção dinâmica é delimitada por um par de *tags*[6]*[% %]*. Qualquer texto externo a esse par de *tags* está associado a uma seção estática[26]. A Figura 3.4 ilustra um exemplo de código escrito em EGL que gera texto no formato HTML.

[5] Um formato predefinido que serve como modelo para criação de documentos ou arquivos.
[6]Conjunto de caracteres que delimita um item de dados a fim de identificá-lo.

example.egl

```
1  [%
2      var repository : Repository := Repository.allInstances().at(0);
3  %]
4
5  <html>
6      <head>
7          <title>Revisions</title>
8      </head>
9      <body>
10         <h1>Revisions</h1>
11         [% for (revision in repository.revisions) { %]
12             [%=revision.name%]
13         [%}%]
14     </body>
15 </html>
16
```

Figura 3.4 - Exemplo de código EGL.

EOL é uma linguagem de programação imperativa para criação, consulta e modificação de modelos EMF. EOL possui uma sintaxe bem semelhante à linguagem *Javascript*[34], fornecendo as características imperativas habituais como presença de variáveis, comandos sequenciais, estruturas de controle e de desvio condicional, além de outras características adicionais advindas do OCL[35].As demais linguagens *Epsilon*, como EGL e EVL, herdam algumas das características de EOL e podem incluir trechos de código EOL em suas definições[30][26].

EWL é uma linguagem utilizada para definição de transformações no modelo através de *wizards*[7]que podem então ser executados a partir do editor gráfico EMF. Um *wizard* consiste de três partes: o guarda (*guard*), o título (*title*) e a ação (*do*). O guarda define os elementos do modeloaos quais o*wizard* é aplicado, o título simplesmente provê um nome ao *wizard* e a ação implementa a lógica do *wizard*[33][26]. A Figura 3.5 mostra um exemplo de *wizard* e seus componentes.

[7]Funcionalidade de um programa que automatiza tarefas complexas.

example.ewl

```
1  wizard ExtractInterface {
2
3      guard : self.isKindOf(Class)
4      title : 'Extract interface from class ' + self.name
5      do {
6          var newName : String;
7          var message : String := 'Enter a name for the interface';
8          newName := UserInput.prompt(message, 'I' + self.name);
9
10         // ...
11     }
12 }
13
```

Figura 3.5 - Exemplo de código EWL.

3.2.Oracle PL/SQL

Oracle é uma tecnologia de sistemas de banco de dados relacional que fornece recursos abrangentes e é facilmente integrada com linguagens de programação, principalmente com Java. PL/SQL significa “Extensões de linguagens procedurais para SQL” em português, o que denota que PL/SQL é totalmente integrado com SQL, além de acrescentar construções de programação que não são nativas de SQL[36][37]. A linguagem PL/SQL tem uma sintaxe própria para construção de rotinas, então é relevante que estas construções sejam estudadas. Nas próximas seções serão abordadas as construções básicas necessárias para a criação de subprogramas usando PL/SQL.

3.2.1. Subprogramas Oracle PL/SQL

Subprogramas ou rotinas são blocos PL/SQL que podem ser chamados com um conjunto de parâmetros. PL/SQL tem dois tipos de subprogramas, procedimentos e funções. Um subprograma PL/SQL é composto basicamente por uma parte declarativacom declarações de tipos, cursores, variáveis; uma parte executávelcom comandos que atribuem valores, controlam a execução e manipulam dados *Oracle;*euma parte opcional para manipulação de exceções.A ferramenta proposta focará na modelagem de procedimentos, mas os conceitos abordados aqui também são válidos para funções.

3.2.2. Procedimentos Oracle PL/SQL

Um procedimento é um subprograma que realiza uma tarefa específica. Procedimentos possuem essencialmente dois componentes: a especificação e o corpo. A especificação começa com a palavra-chave PROCEDURE e contém o nome e a lista de parâmetros (que pode ser vazia) da rotina. O corpo aparece logo após a lista de parâmetros e é iniciado com a palavra-chave IS (ou AS) e finalizado com END. Dentro do corpo há três partes: uma parte declarativa opcional, uma parte executável, e uma parte de manipulação de exceções também opcional (esta última parte não será tratada neste trabalho). A parte declarativa contém declaração de tipos, cursores, constantes, variáveis, entre outros. A parte executável contém comandos que atribuem valores, controlam a execução e manipulam dados Oracle[36]. A Figura 3.6 ilustra a sintaxe de um procedimento Oracle PL/SQL.

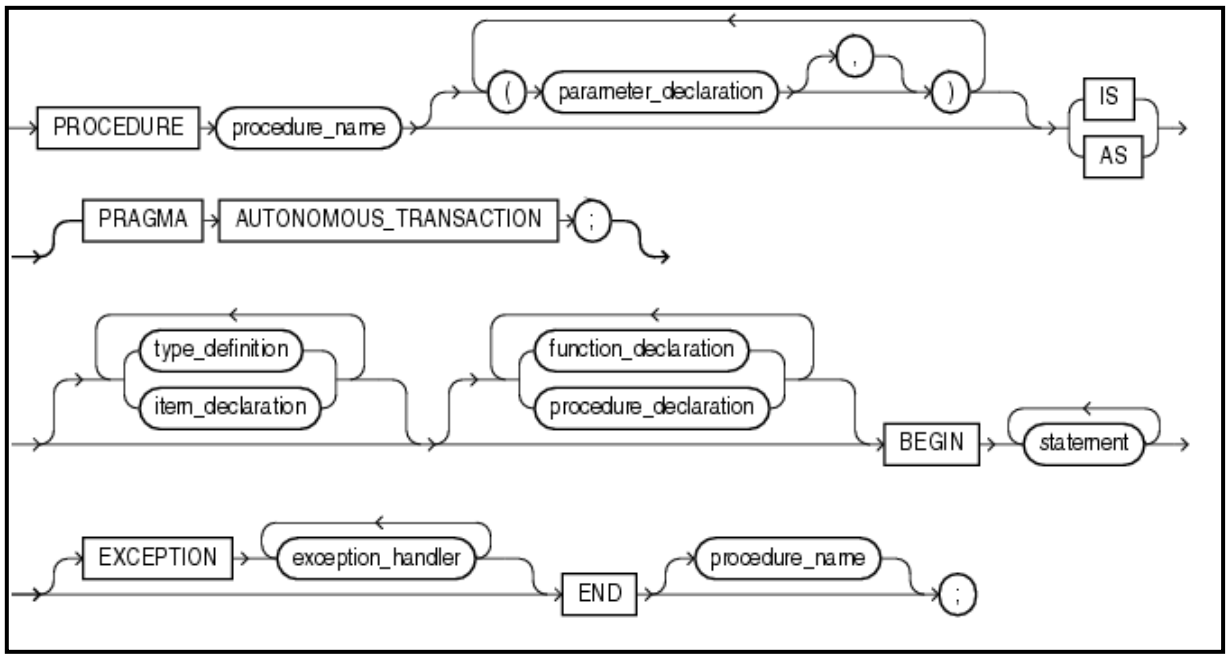

Figura 3.6 - Sintaxe de um procedimento Oracle PL/SQL (Oracle® Database PL/SQL User'sGuideandReference)

Para cada parâmetro, é preciso especificar o nome, o modo (IN, OUT ou IN OUT), o tipo de dados Oracle e um valor padrão opcional. O modo determina o comportamento de cada um dos parâmetros. Parâmetros com modo IN passam valores para o subprograma sendo chamado, enquanto que parâmetros com modo OUT retornam valores ao programa que chama o

subprograma. Parâmetros IN OUT tanto passam valores ao procedimento como retornam valores atualizados [36]. Há vários tipos de dados nativos de Oracle e cada um deles define um formato específico de armazenamento, restrições e um intervalo válido de valores. Os tipos de dados mais comumente utilizados no Oracle são NUMBER para valores numéricos, VARCHAR2 e CHAR para cadeias de caracteres, e DATE e TIMESTAMP para datas e horários[37].Existe um tipo de dados em Oracle chamado SYS_REFCURSOR que permite que um conjunto de registros seja retornado por uma rotina. O Anexo B provê um quadro com todos os tipos de dados predefinidos em Oracle. A Figura 3.7ilustra a declaração de um parâmetro usando Oracle PL/SQL.

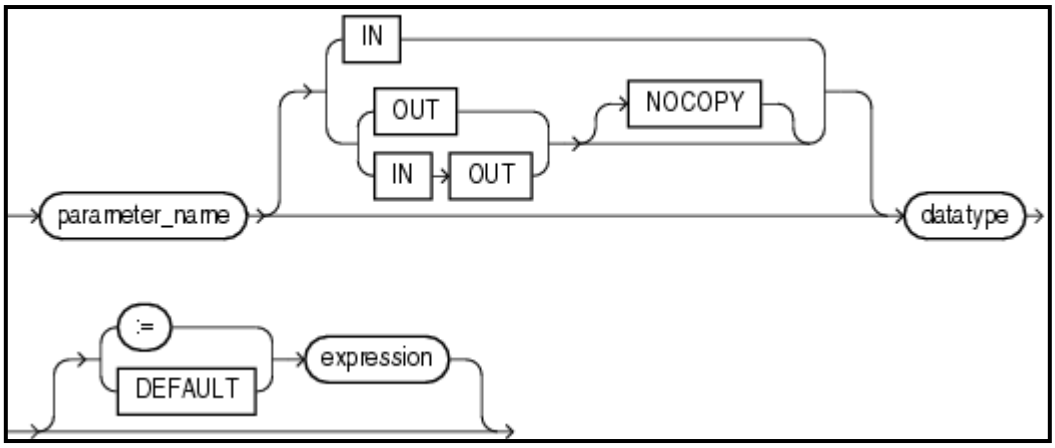

Figura 3.7 – Sintaxe de declaração de um parâmetro em Oracle PL/SQL (Oracle® Database PL/SQL)

Declarações de variáveis devem ser feitas na parte declarativa do subprograma PL/SQL. Declarações alocam espaço de armazenamento para um valor, especificam seu tipo de dados, seu identificador (nome) e valor inicial (opcional)[36]. Existem declarações de variáveis que utilizam os qualificadores especiais %TYPE e %ROWTYPE em suas definições. O atributo %TYPE provê o tipo de dados de uma variável ou coluna de tabela. O atributo %ROWTYPE provê um tipo registro que representa uma linha em uma tabela.A Figura 3.8ilustra um exemplo de trecho de código com declarações de variáveis. Nas linhas 1 e 2, as variáveis são declaradas e inicializadas. O operador de atribuição é representado pelo símbolo de dois pontos seguido por um sinal de igual (:=).

```
1   name VARCHAR2(64) := 'André';
2   pi NUMBER := 3.1415926;
3   birth DATE;
4   person_id PERSON.id%TYPE;
5   person_record PERSON%ROWTYPE;
```

Figura 3.8 - Exemplo de código PL/SQL com declarações de variáveis

Um cursor é um mecanismo pelo qual é possível atribuir um nome a um comando SELECT e manipular as informações dentro dessa instrução SQL. Declarações de cursores são compostas por um nome e uma consulta específica associada[36],[37]. A Figura 3.9 ilustra um exemplo de trecho de código com declaração de cursor.

```
1   CURSOR cursor_name
2   IS
3       SELECT person_id, person_name
4         FROM person_table
5         WHERE person_address = v_address;
```

Figura 3.9 - Exemplo de código PL/SQL com declaração de cursor

A parte executável de um subprograma PL/SQL é composta por comandos de atribuição e manipulação de dados e estruturas de controle[36].Como foi visto anteriormente, no momento da declaração, uma maneira atribuir valores a uma variável é usando o operador de atribuição (:=). A linha 1 da Figura 3.10 ilustra um exemplo de utilização do operador de atribuição.Outra forma de atribuir valores a uma variável é selecionando valores armazenados na base de dados usando o comando SELECT INTO, como é ilustrado na linha 2 da Figura 3.10.

```
1   country := 'France';
2   SELECT salary * 0.10 INTO bonus FROM employees
3       WHERE employee_id = emp_id;
```

Figura 3.10 - Exemplo de código PL/SQL com atribuições

Uma das estruturas de controle mais importantes é o IF. Este comando condicional executa ou ignorauma sequência de outros comandos de acordo com o valor de uma condição. Sua forma mais simples é o IF-THEN, no qual uma sequência de comandos é executada apenas

se a condição for verdadeira. A segunda forma do comando IF (IF-THEN-ELSE) adiciona a palavra-chave ELSE seguida por uma sequência alternativa de comandos que são executados caso a condição seja falsa ou nula. Quando é preciso escolher entre várias alternativas, usa-se a palavra-chave ELSIF para introduzir condições adicionais. As condições são testadas uma por uma de cima para baixo. Se uma dada condição é verdadeira, o bloco de instruções associado a ela deve ser executado. Se nenhuma das condições for verdadeira, o bloco correspondente ao ELSE (o último) é executado[36]. A Figura 3.11 ilustra a sintaxe do comando IF.

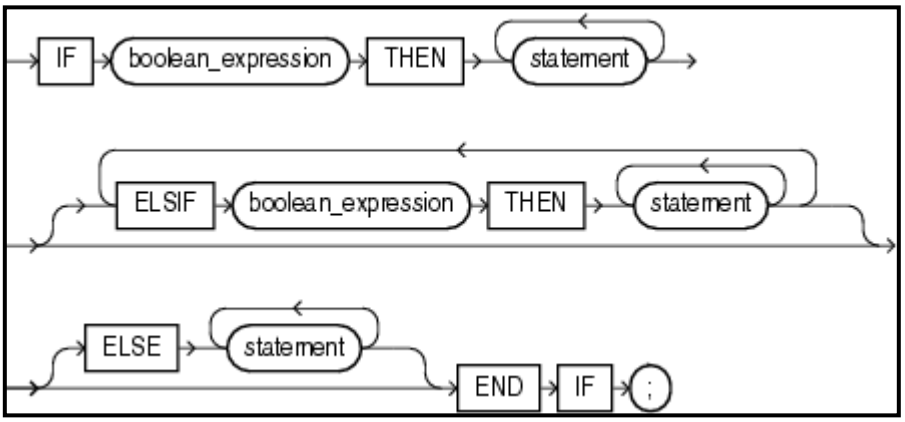

Figura 3.11 - Sintaxe do comando IF (Oracle® Database PL/SQL User'sGuideandReference)

No Oracle PL/SQL há várias estruturas de repetição que executam uma seqüência de comandos repetidamente enquanto a condição de parada não for estabelecida. A forma mais simples é o laço básico que inclui uma seqüência de comandos entre as palavras-chaves LOOP e END LOOP. Existem dois comandos que servem para parar a execução contínua do laço, evitando loop infinito. Um deles força o laço a parar incondicionalmente, é o comando EXIT. O outro, o comando EXIT-WHEN, para o laço quando a condição na cláusula WHEN for verdadeira, ou seja, de forma condicional. Laços simples também podem conter um rótulo opcional com identificadores associados que podem ser referenciados pelos comandos de saída[36],[37]. A Figura 3.12 ilustra a sintaxe do comando LOOP.

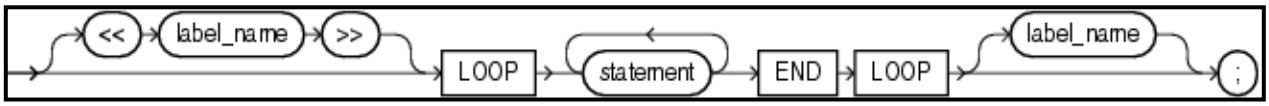

Figura 3.12 - Sintaxe do comando LOOP (Oracle® Database PL/SQL User'sGuideandReference)

Os outros comandos de repetição são o WHILE-LOOP e o FOR-LOOP. O WHILE-LOOP executa os comandos que estão dentro do corpo do laço enquanto a condição é verdadeira. A condição é avaliada antes de cada iteração. Se a condição for verdadeira, a seqüência de comandos é executada, caso contrário, o laço é ignorado e o controle é passado para o próximo comando[36],[37]. A Figura 3.13 ilustra a sintaxe do comando WHILE-LOOP.

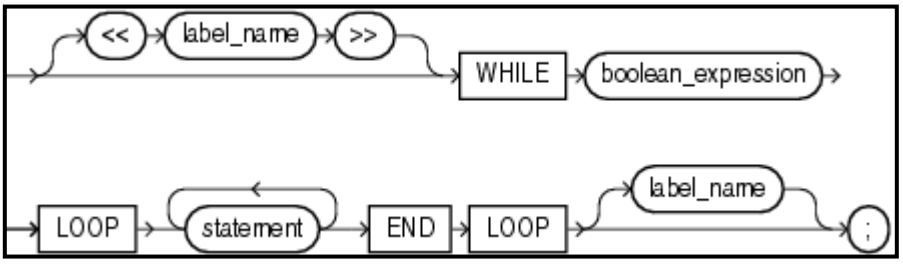

Figura 3.13 - Sintaxe do comando WHILE-LOOP (Oracle® Database PL/SQL User'sGuideandReference)

O comando FOR-LOOP itera sobre um intervalo especificado de números inteiros, portanto, o número de iterações é conhecido antecipadamente. O operador simbolizado por um ponto duplo (..) fica posicionado entre o limite inferior e superior, indicando o intervalo. Por padrão, a iteração prossegue ascendentemente, partindo do limite inferior até o limite superior, mas a palavra-chave REVERSE pode ser usada para indicar variação descendente do intervalo[36]. A Figura 3.14 ilustra a sintaxe do comando FOR-LOOP.

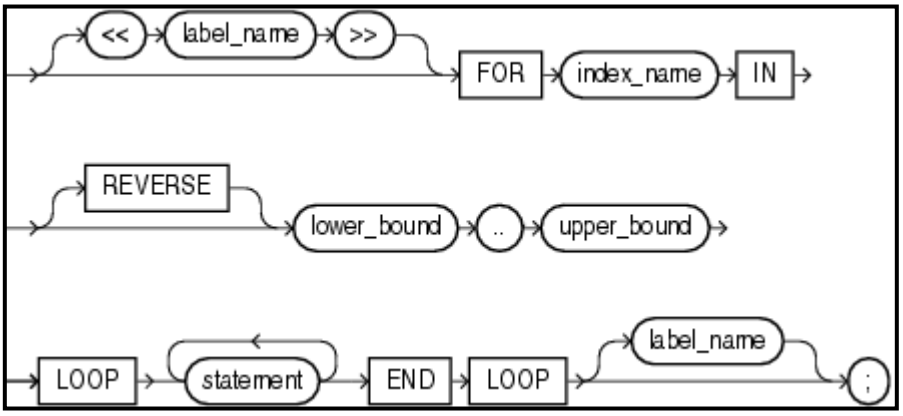

Figura 3.14 - Sintaxe do comando FOR-LOOP (Oracle® Database PL/SQL User'sGuideandReference)

O próximo capítulo se dedica aos detalhes do protótipo desenvolvido, mostrando seus recursos e funcionalidades disponíveis.

4. A FERRAMENTA

Nesta seção será apresentada a ferramenta desenvolvida neste trabalho. Esta ferramenta tem como propósito a modelagem de procedimentos a partir de elementos gráficos.A principal motivação para o seu desenvolvimento foi suprir as limitações encontradas nas ferramentas analisadas na seção 2, focando na sua aplicação juntamente com ferramentas de geração de relatórios. As ferramentas referidas permitiam apenas a definição do "esqueleto" do procedimento, não dando suporte à especificação do corpo da rotina.

Na seção 4.1 serão apresentadas as tecnologias propostas para o desenvolvimento da ferramenta, na seção 4.2 será apresentado o seu funcionamento básico e na seção 4.3 serão apresentados os componentes que compõem sua interface gráfica.

4.1.Tecnologias Propostas

Diante do estudo realizado ao longo deste trabalho, as tecnologias para o desenvolvimento de uma ferramenta gráfica para modelagem de procedimentos são:

- *Eclipse Modeling Project*
  - *Framework*EMF
  - *Framework*GMF
  - Plataforma *Epsilon*
- Linguagem de programação: *Java*
- Solução de banco de dados: *Oracle 10g*

O *Eclipse Modeling Project* compreende um conjunto unificado de frameworks de modelagem, ferramentas e implementações que facilitam bastante a tarefa de desenvolvimento de ferramentas CASE, por isso a sua escolha. A escolha de Java como linguagem de programação está diretamente relacionada com as ferramentas descritas anteriormente. O protótipo da ferramenta será compatível primeiramente apenas com subprogramas Oracle PL/SQL, gerando código para tal linguagem-alvo,podendo ser posteriormente estendido para outras linguagens.

4.2.Funcionamento da Ferramenta

O protótipo desenvolvido é um plug-in para o Eclipsee permite a criação de procedimentos Oracle PL/SQL modelando-os em um diagrama. O diagrama representa o procedimento e as construções da linguagem são representadas por elementos gráficos (retângulos). Estes elementos podem ser adicionados ao diagrama, uns dentro dos outros, formando a rotina.

Uma vez modelado o procedimento, o código referente a ele pode ser gerado. Através de uma conexão ao sistema de banco de dados Oracle, o código gerado pode ser armazenado no servidor para posterior execução.A ferramenta, então, consiste de duas principais funcionalidades: modelagem e geração do código de um procedimento Oracle PL/SQL.

4.3.Interface Gráfica

Como a ferramenta desenvolvida é um plug-in do Eclipse, esta utiliza todo o ambiente de desenvolvimento que o mesmo dispõe, podendo ser integrada a ele sem a necessidade de instalação de um novo *software*. A Figura 4.1 ilustra a interface gráfica do protótipo.

4.3.1. Ambiente do *Eclipse*

Do lado esquerdo da interface, há uma aba chamada *Project Explorer*(1) que provê uma visualização hierárquica dos recursos presentes no espaço de trabalho (*Workspace*). É a partir do *Project Explorer* que o usuário poderá visualizar e editar os diagramas existentes e criar novos.

A área central do programa fica o *Editor Area* (2), o local em que o usuário modifica o conteúdo dos arquivos do *Workspace*. Este é o espaço em que editor gráfico fica localizado, permitindo a modelagem do procedimento.

Na área interna do *Editor Area* fica um componente chamado *Palette* (3). As ferramentas pertencentes à paleta podem ser usadas para seleção, criação de objetos e qualquer outra ação a ser realizada no diagrama.

Na parte inferior do programa é possível encontrar a aba chamada *Properties* (4) que mostra as propriedades básicas de um recurso selecionado. Esta aba de propriedades permite que o usuário visualize e configure os valores das propriedades de cada elemento do diagrama.

Na parte inferior esquerda do programa está localizada a aba *Outline*(5) que apresenta um esboçodo arquivo atualmente aberto no *Editor Area*e lista seus elementos estruturais.

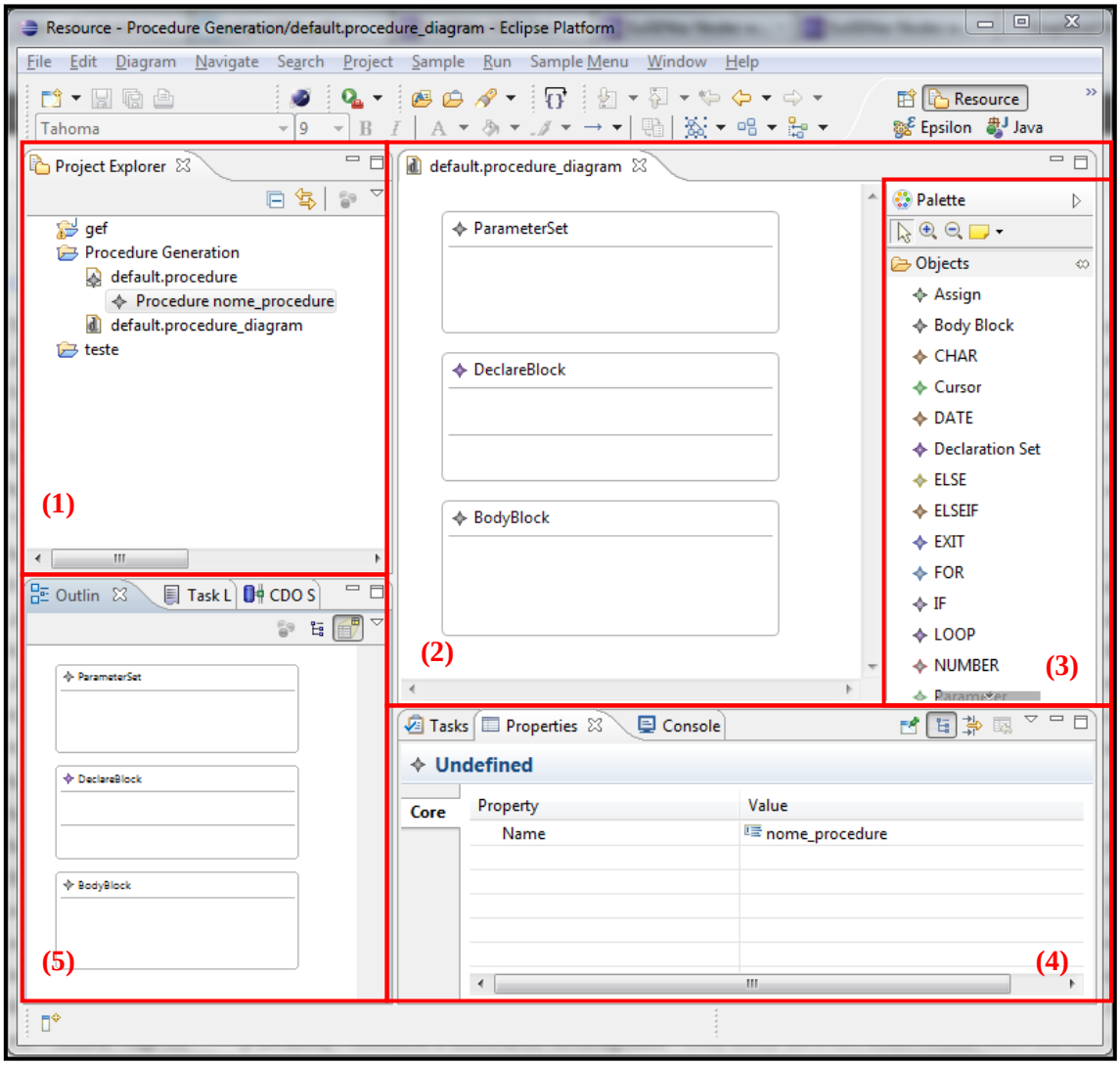

Figura 4.1 - Interface gráfica do protótipo: *PackageExplorer* (1), *EditorArea* (2), *Palette* (3), *Properties* (4) e *Outline* (5).

4.3.2. Geração do código do procedimento

Com o diagrama do procedimento modelado, é possível gerar o código fonte Oracle PL/SQL associado. Para isso, existe um botão na barra de ferramentas que quando clicado abre uma janela em que o usuário pode visualizar e salvar o código gerado em um arquivo do sistema e executar o procedimento, armazenando-o no banco de dados para ser usado posteriormente.NaFigura 4.2é possível ver a localização do botão*Generate Procedure*.

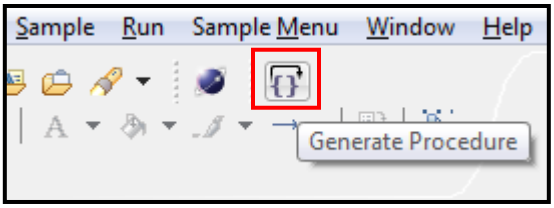

Figura 4.2 - Localização do botão para geração de código do procedimento.

A janela que se abre ao clicar no botão *Generate Procedure* é ilustrada na Figura 4.3. Através da interface é possível se conectar uma instância do Oracle para a visualização dos procedimentos atualmente armazenados e para o armazenamento do código referente ao procedimento recém modelado. O código gerado pode ser simplesmente copiado ou salvo em algum arquivo do sistema.

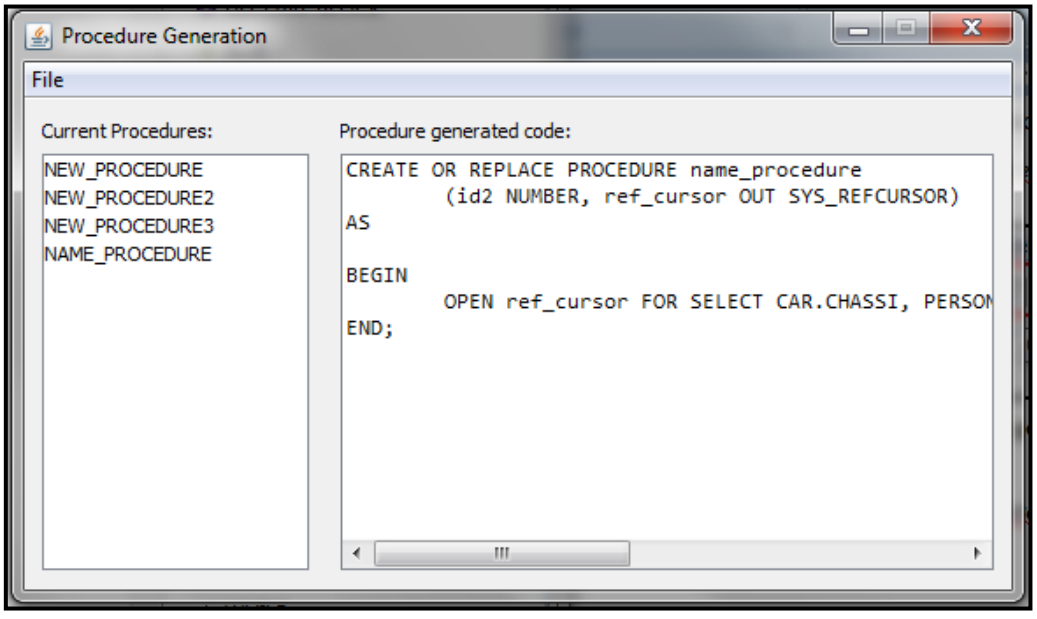

Figura 4.3 - Janela para geração do código do procedimento (*Procedure Generation*).

4.3.3. Criação de consultas SQL

A ferramenta conta com um *Query Builder* que permite ao usuário criar consultas SQL via interface gráfica e utilizá-las na declaração e inicialização de cursores, por exemplo. A Figura 4.4 ilustra a interface gráfica para criação de consultas. O usuário pode selecionar colunas de tabelas armazenadas no banco de dados,atribuir apelidos (*aliases*) e definir condições simples para estes.

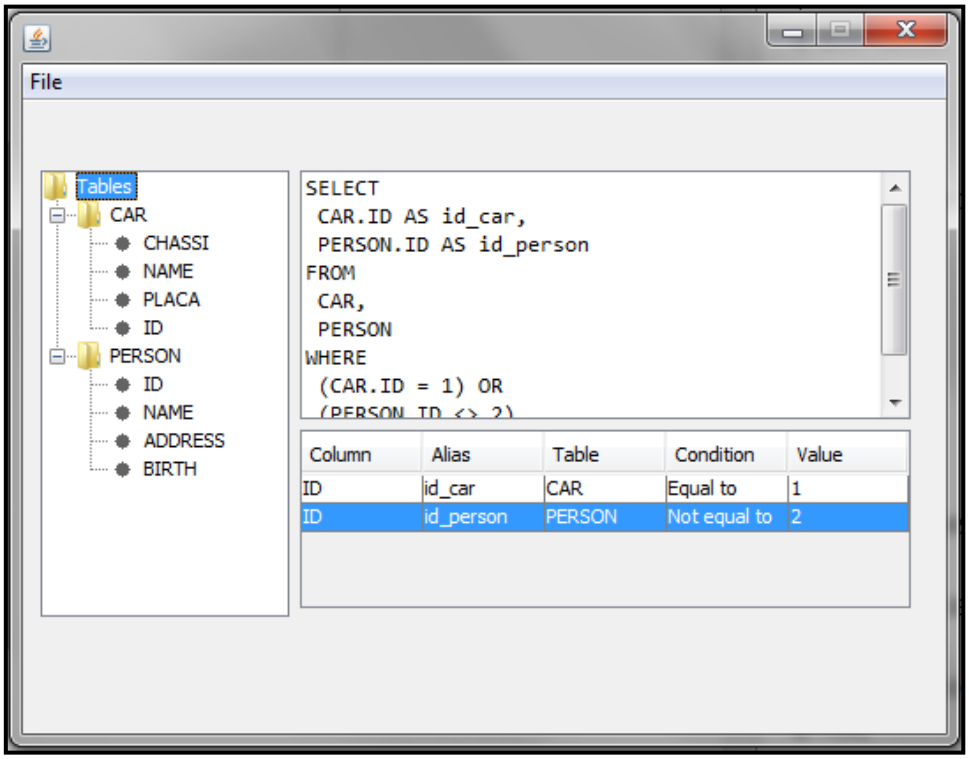

Figura 4.4 - Interface gráfica para criação de consultas SQL.

4.3.4. Interface para seleção de colunas e tabelas

Outro componente do sistema é a interface gráfica que permite a escolha de uma coluna ou linha da tabela associada a variáveis do tipo TYPE ou ROWTYPE, respectivamente. A interface é ilustrada na Figura 4.5.

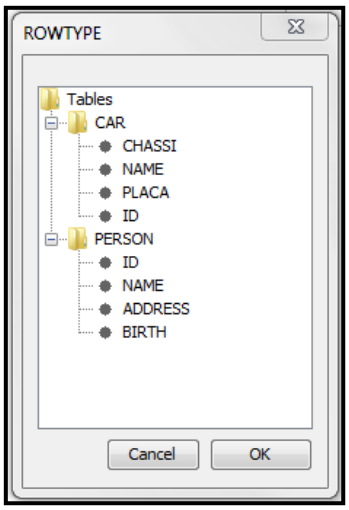

Figura 4.5 - Interface para escolha de coluna/linha da tabela.

4.3.5. Diagrama de modelagem

O principal componente do sistema é o diagrama que permite modelar o procedimento. A representação gráfica do procedimento no editor gráfico é dividida em três componentes principais que são ilustrados na Figura 4.6:

- ***Parameter Set*** – Representa o conjunto de parâmetros do procedimento*;*
- ***Declare Block***–Bloco de declarações de variáveis, tipos e cursores do procedimento;
- ***BodyBlock*** – Corpo do procedimento.

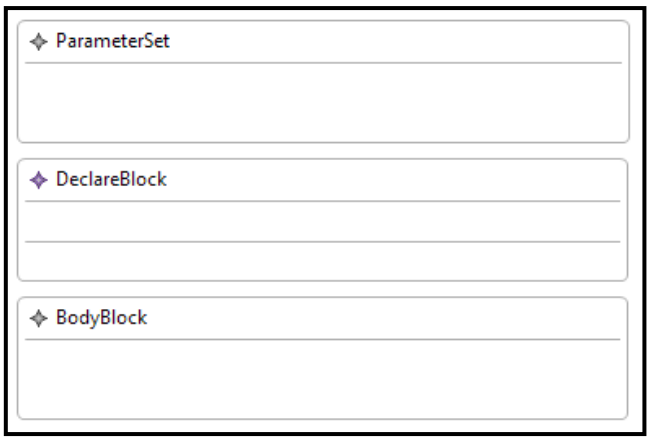

Figura 4.6 - Os três componentes principais do diagrama: Parameter Set, Declare Block e BodyBlock.

Dentro do bloco*Parameter Set*é possível adicionar elementos do tipo *Parameter* e *RefCursor*como ilustrado naFigura 4.7. Os elementos do tipo *Parameter*(1) representam parâmetros com tipos de dados escalares NUMBER, CHAR, VARCHAR2 e DATE. Elementos do tipo *RefCursor* (2) representam parâmetros com o tipo de dado SYS_REFCURSOR. Além de definir o tipo de dado do parâmetro, é possível configurar o seu identificador e o seu comportamento (IN, OUT, IN OUT ou padrão).

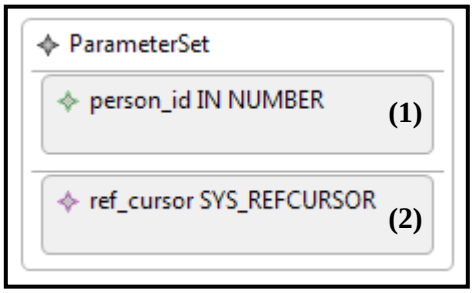

Figura 4.7 - Representação dos elementos *Parameter*(1) e *RefCursor* (2).

Dentro do bloco *Declare Block*é possível adicionar elementos que representam declarações de variáveis de tipos predefinidos (*Number*, *Varchar2*, *Char*, *Date*), declaração de cursores (*Cursor*) e declarações de variáveis de colunas ou linhas de tabelas (*Type* e *RowType*, respectivamente). Um exemplo de configuração deste bloco é ilustrada na Figura 4.8. As declarações de variáveis de tipos predefinidos requerem basicamente a configuração de um identificador, há algumas propriedades adicionais para definição da precisão e escala numérica (no caso do tipo *Number*), e o tamanho (no caso dos tipos *Varchar2* e *Char*). O tipo *Cursor* possui um nome e uma consulta SQL associada que pode ser gerada pelo *Query Builder*. Os tipos *Type* e *RowType* possuem um identificador e uma tabela associada. No caso do tipo *Type*, também há um campo para o nome da coluna da tabela. Os campos tabela e coluna podem ser definidos via interface gráfica.

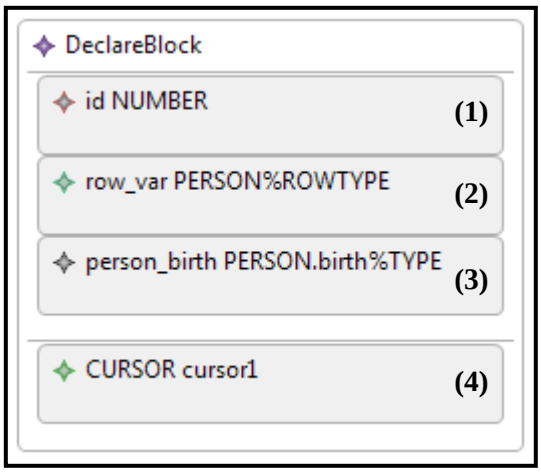

Figura 4.8 - Representação dos elementos *Number* (1), *RowType* (2), *Type* (3) e *Cursor* (4).

O bloco *BodyBlock* agrupa os tipos que representam os comandos de atribuição (*Assign*), controle da execução (*If*, *ElseIf*, *Else*, *Loop*, *ExitLoop*, *While* e *For*) e manipulação de dados (*OpenRefCursor*). A Figura 4.9 ilustra o *BodyBlock* e seus componentes. O tipo *Assign*(1) representa uma atribuição de uma expressão a uma variável. Os tipos *If* (2), *ElseIf* (3) e *Else* (4) representam o comando condicional IF de PL/SQL. Todos os três tipos podem conter outros comandos em seu interior. Os tipos*If* e *ElseIf* possuem um atributo para definição da condição booleana. Os tipos *Loop*, *While* e *For* representam as estruturas de repetição LOOP, WHILE-LOOP e FOR-LOOP, respectivamente, e podem conter outros comandos internos. O tipo *Loop* possui um atributo opcional para definição do seu rótulo. O tipo *While* possui apenas um atributo referente à condição booleana. O tipo *For* possui um atributo para definir o nome do contador do loop, dois atributos para delimitar o intervalo (limite inferior e superior) e um atributo booleano para indicar variação crescente ou decrescente do valor do contador. O tipo *OpenRefCursor*possui um REF CURSOR e uma consulta SQL associada.

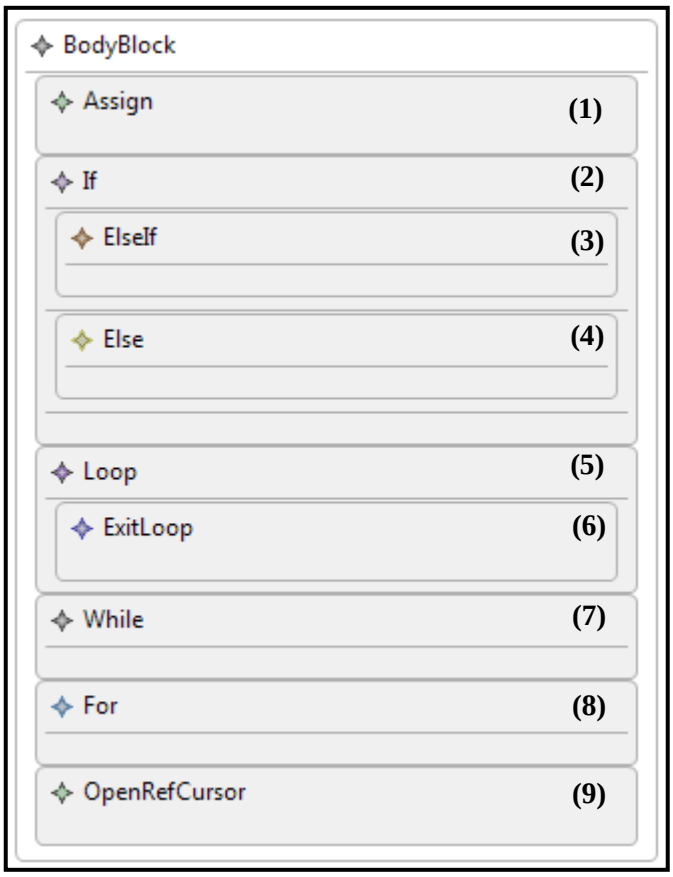

Figura 4.9 – Representação dos elementos *Assign* (1), *If* (2), *ElseIf* (3), *Else* (4), *Loop* (5), *ExitLoop* (6), *While* (7), *For* (8) e *OpenRefCursor* (9).

O próximo capítulo se dedica a apresentar a utilização da ferramenta desenvolvida juntamente com uma ferramenta de geração de relatórios.

5. ESTUDO DE CASO

Este capítulo tem como objetivo mostrar a utilização do protótipo desenvolvido durante este trabalho em conjunto com uma ferramenta de geração de relatórios (*Reporting Tool*).Na Seção 5.1 será apresentada a ferramenta para geração de relatórios utilizada e na seção 5.2 será detalhado o procedimento realizado no estudo.

5.1.Eclipse BIRT

A ferramenta escolhida para utilização juntamente com o protótipo desenvolvido foi o Eclipse BIRT (*Business IntelligenceandReporting Tools*). O Eclipse BIRT é um projeto código aberto baseado no Eclipse que permite a produção de relatórios a partir de dados provenientes de diversos tipos de fontes, incluindo banco de dados.O sistema ainda fornece exportação do relatório para diversos formatos como HTML, PDF, XLS e DOC[3]. A Figura 5.1ilustra a interface gráfica do Eclipse BIRT.

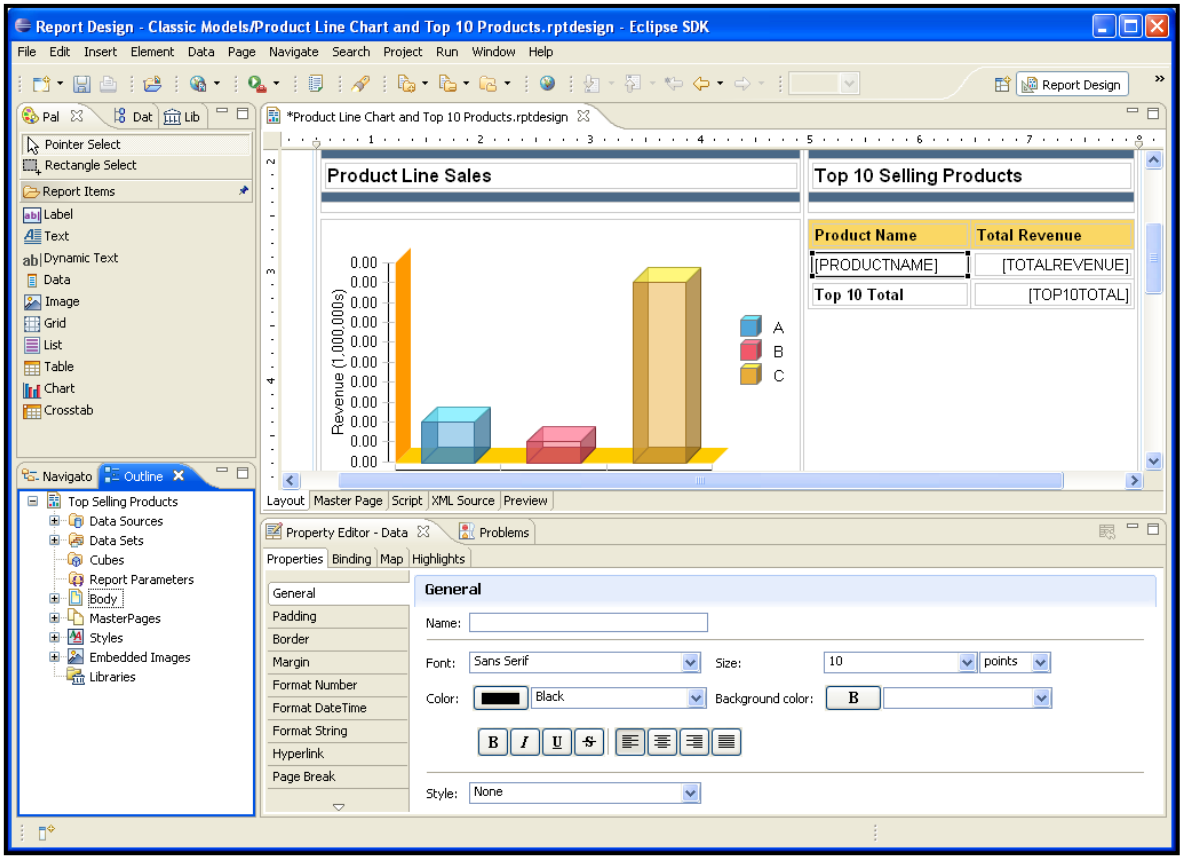

Figura 5.1- Interface gráfica do programa *Eclipse* BIRT.

5.2.Métodoutilizado

O método utilizado foi simples. Primeiramente, o procedimento foi criado e armazenado no banco de dados utilizando o protótipo desenvolvido. Depois, um relatório foi criado utilizando o Eclipse BIRT chamando o procedimento anteriormente citado.

Uma tabela foi utilizada para consulta dos dados. A tabela PERSON que possui as colunas *id*, *name* e *age* que representam o identificador, nome e idade de uma pessoa, respectivamente.A Figura 5.2 ilustra a tabela.

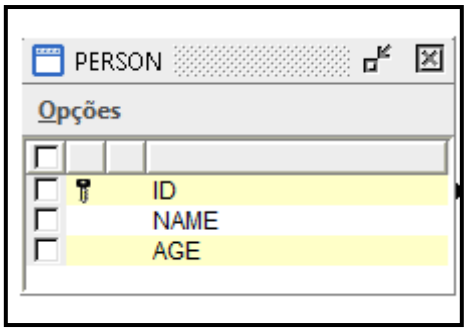

Figura 5.2 - Tabela PERSON.

O objetivo era recuperar o identificador, nome e idade das pessoas que possuem idade maior que um determinado número.Sendo assim, o procedimento modelado foi simples, possuindo um parâmetro de entrada do tipo NUMBER e um parâmetro de saída do tipo SYS_REFCURSOR para retorno da consulta. Foi necessário adicionar o comando OPEN FOR ao corpo do procedimento para associar o parâmetro de saída com a consulta apropriada. Não foi preciso declarar variáveis. O diagrama modelado está ilustrado na Figura 5.3.

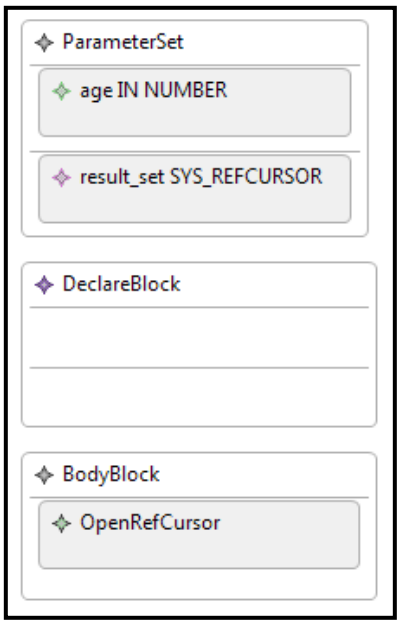

Figura 5.3 - Diagrama do procedimento modelado.

Para criar a consulta SQL desejada e associá-la ao parâmetro de saída, a interface gráfica do *Query Builder* do protótipo foi utilizada. A Figura 5.4 ilustra a criação da consulta.

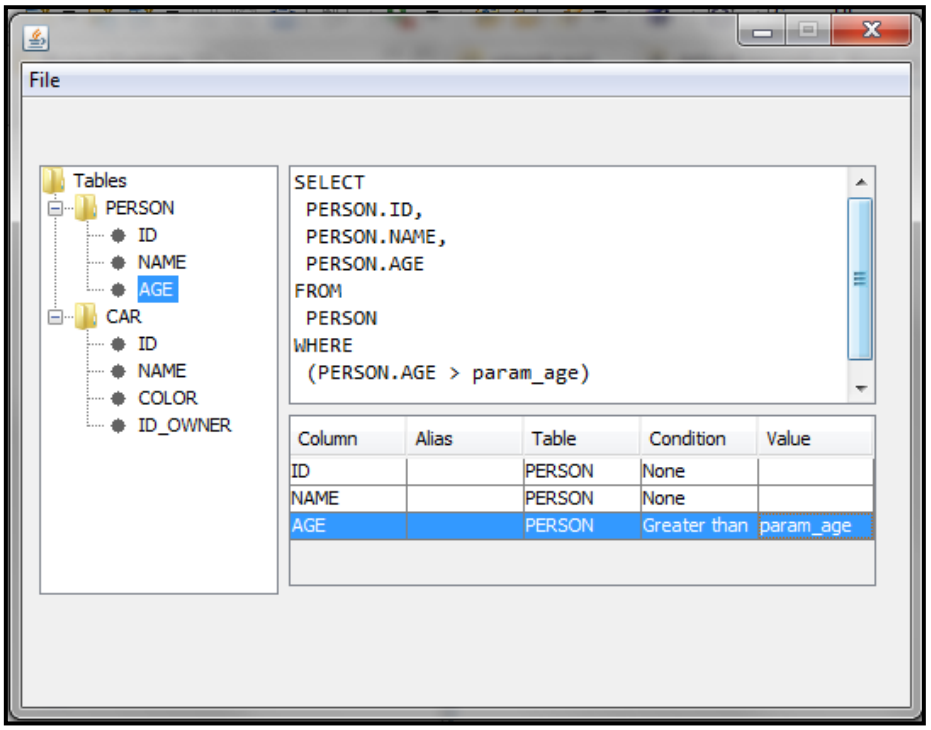

Figura 5.4 - Utilização do *QueryBuilder* para criação da consulta.

Com o procedimento já modelado, o código associado foi gerado com auxílio da interface do *Procedure Generation* conforme mostrado na Figura 5.5. Após isso, o procedimento foi armazenado no banco de dados para ser chamado através do *Eclipse* BIRT. O código gerado pela ferramenta é mostrado na Figura 5.6.

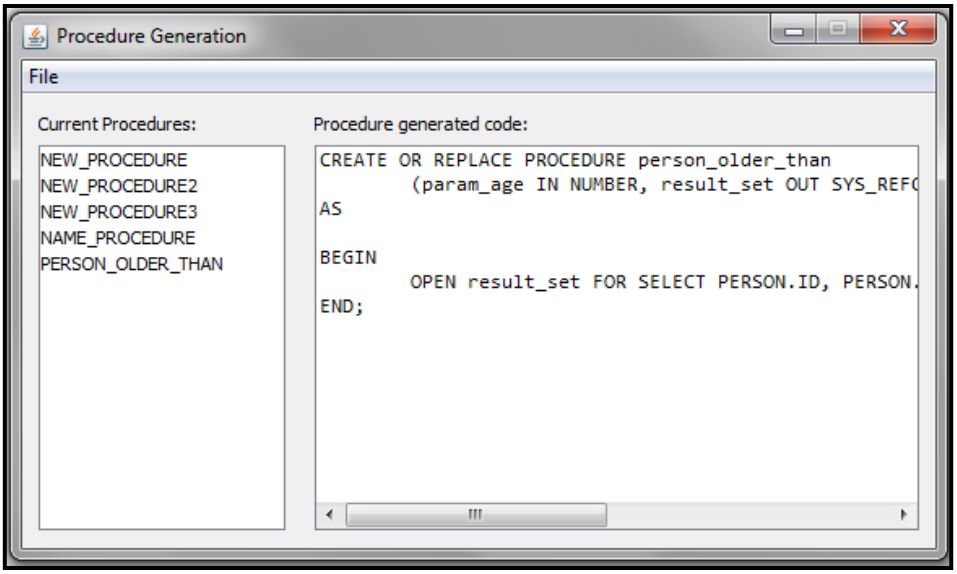

Figura 5.5 - Geração do código do procedimento e armazenamento no banco de dados.

```
CREATE OR REPLACE PROCEDURE person_older_than
    (param_age IN NUMBER, result_set OUT SYS_REFCURSOR)
AS

BEGIN
    OPEN result_set FOR
        SELECT PERSON.ID, PERSON.NAME, PERSON.AGE
        FROM PERSON
        WHERE (PERSON.AGE > param_age) ;
END;
```

Figura 5.6 - Código do procedimento gerado.

Para a criação do relatório utilizando o *Eclipse* BIRT, primeiramente foi definido um *Data Source* que é simplesmente uma conexão com uma fonte de dados, neste caso, o banco de dados *Oracle*. A partir do *DataSource* configurado, foi criado um *Data Set* que representa um conjunto de dados recuperado. É possível definir um conjunto de dados a partir de uma consulta

SQL ou chamando rotinas, como foi realizado. Através da interface do BIRT também é permitido definir quais os valores são passados para o procedimento chamado. No caso do exemplo aqui exposto, foi definido o valor 21 para o parâmetro de entrada, o que significa que o conjunto de dados retornado inclui as pessoas com idade maior que 21.

Utilizando o *ReportDesigner* do BIRT, foi criado um simples relatório com um título e uma tabela para visualização dos dados. O relatório gerado pode ser visto na Figura 5.7.

PEOPLE WHO ARE OLDER THAN 21

ID	NAME	AGE
1	André Ricardo	22
3	José Jorge	50
4	Socorro Lopes	52
5	Thaís Melise	22
6	Evelyn Mirella	24

Figura 5.7 - Relatório gerado pelo Eclipse BIRT chamando o procedimento criado.

O estudo apresentado neste capítulo mostrou a possibilidade de utilizar procedimentos modelados com a ferramenta proposta na geração de relatórios, em conjunto com uma ferramenta apropriada.

O próximo capítulo traz as principais contribuições, dificuldades e trabalhos futuros para o presente trabalho.

6. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este capítulo mostra as conclusões sobre este trabalho. A Seção 6.1 descreve suas principais contribuições. A Seção 6.2 aponta as maiores dificuldades encontradas. Finalmente, a Seção 6.3 indica as perspectivas de trabalhos futuros.

6.1.Contribuições

Os estudos realizados neste trabalho mostraram a deficiência das ferramentas existentes atualmente em relação ao suporte na criação de rotinas de linguagens de quarta geração. As ferramentas exploradas apenas forneciam um modo para definição do cabeçalho da rotina, não fornecendo qualquer suporte à definição do seu corpo.A ferramenta proposta neste trabalho supriu tal deficiência. Com o seu uso, é possível modelar um procedimento completo e não apenas sua estrutura, definir parâmetros, declarações e comandos, e aplicá-lo na geração de relatórios.

6.2.Dificuldades Encontradas

A tarefa de modelar um procedimento utilizando elementos gráficos não é fácil. Talvez esse tenha sido o motivo da inexistência de uma ferramenta deste tipo no mercado. Um fator complicador foia determinação do escopo, o que seria levado em consideração ou não no desenvolvimento do protótipo, quais construções de PL/SQL deveriam ser incluídas. Outro empecilho foi achar uma maneira boa de representar as construções da linguagem de forma gráfica no diagrama.Além disso, problemas no desenvolvimento da interface gráfica, em como customizá-la para melhor utilização do usuário.

6.3.Trabalhos Futuros

Uma continuidade interessante deste trabalho seria a modelagem das construções do procedimento de tal forma que este fosse compatível com rotinas de outros sistemas de gerenciamento de bancos de dados, além do Oracle, a fim de ter uma utilidade mais abrangente.

Adaptar a ferramenta para permitir a criação de funções, além de procedimentos. Como as duas rotinas são bem similares, o trabalho de adaptação seria mínimo.

Permitir a realização de chamadas a subprogramas externos dentro da rotina, além da declaração e manipulação de tipos complexos (como registros, *varrays* e *nestedtables*) e outras operações de manipulações de dados que não foram implementadas.

Enriquecer o diagrama com funcionalidades que aperfeiçoariam a experiência do usuário durante a tarefa de modelagem do procedimento: construções poderiam estar dispostas de forma sequencial, os elementos poderiam ser movidos de localização, facilitando reuso, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

1. GREGORY HILL. Enterprise Reporting Guide. Disponivel em: <http://ghill.customer.netspace.net.au/reporting/>. Acesso em: 01 nov. 2011.

2. WORLD WIDE WEB CONSORTIUM (W3C). Disponivel em: <http://www.w3.org/XML/>. Acesso em: 01 dez. 2011.

3. ECLIPSE FOUNDATION. **Eclipse BIRT Overview**. Disponivel em: <http://www.eclipse.org/birt/intro/>. Acesso em: 01 dez. 2011.

4. RUSSELL, G. SQL – Simple Queries (Napier University - Edinburgh). Disponivel em: <http://db.grussell.org/slides/sql1.pdf>. Acesso em: 01 dez. 2011.

5. ADOBE. **Arquivos PDF | Portable Document Format (PDF) da Adobe – Acrobat**. Disponivel em: <http://www.adobe.com/br/products/acrobat/adobepdf.html>. Acesso em: 01 dez. 2011.

6. MICROSOFT. **Microsoft Excel 2010 - Office.com**. Disponivel em: <http://office.microsoft.com/pt-br/excel/>. Acesso em: 01 dez. 2011.

7. MICROSOFT. Microsoft Word 2010 – Office.com. Disponivel em: <http://office.microsoft.com/pt-br/word/>. Acesso em: 01 dez. 2011.

8. WORLD WIDE WEB CONSORTIUM (W3C). **W3C XHTML2 Working Group Home Page**. Disponivel em: <http://www.w3.org/MarkUp/>. Acesso em: 01 dez. 2011.

9. MENDES, W. Linguagem de programação de quarta geração. **Knol**. Disponivel em: <http://knol.google.com/k/william-mendes/linguagem-de-programação-de-quarta/28wtpufocpbs6/3>. Acesso em: 01 dez. 2011.

10. HEERING, J.; MERNIK, M.; M. SLOANE, A. **When and how to develop domain-specific languages**. ACM Computing Surveys. New York: [s.n.]. 2005. p. 316–344.

11. CHANG, Y.-M.; ULLMAN, J. Using Oracle PL/SQL. **The Stanford University InfoLab**. Disponivel em: <http://infolab.stanford.edu/~ullman/fcdb/oracle/or-plsql.html>. Acesso em: 01 dez. 2011.

12. ORACLE CORPORATION. Oracle SQL Developer. **ORACLE**. Disponivel em: <http://www.oracle.com/technetwork/developer-tools/sql-developer/overview/index.html>. Acesso em: 30 nov. 2011.

13. ORACLE CORPORATION. What is SQL Developer? Disponivel em: <http://www.oracle.com/technetwork/developer-tools/sql-developer/what-is-sqldev-093866.html>. Acesso em: 30 nov. 2011.

14. ORACLE CORPORATION. MySQL Workbench. Disponivel em: <http://www.oracle.com/us/products/mysql/mysql-workbench-066221.html>. Acesso em: 30 nov.

2011.

15. ECLIPSE FOUNDATION. Eclipse Modeling Project. Disponivel em: <http://eclipse.org/modeling/>. Acesso em: 20 nov. 2011.

16. ECLIPSE FOUNDATION. Eclipse Modeling - EMF - Home. Disponivel em: <http://www.eclipse.org/modeling/emf/>. Acesso em: 20 nov. 2011.

17. ECLIPSE FOUNDATION. Graphical Modeling Framework. Disponivel em: <www.eclipse.org/gmf>. Acesso em: 20 nov. 2011.

18. ECLIPSE FOUNDATION. Epsilon. Disponivel em: <http://eclipse.org/gmt/epsilon/>. Acesso em: 21 nov. 2011.

19. ORACLE CORPORATION. History of Java Technology. Disponivel em: <http://www.oracle.com/technetwork/java/javase/overview/javahistory-index-198355.html>. Acesso em: 01 dez. 2011.

20. STEINBERG, D. et al. **EMF:** Eclipse Modeling Framework. 2º Edição. ed. [S.l.]: Addison-Wesley Professional, 2008.

21. ECLIPSE FOUNDATION. GEF. Disponivel em: <http://www.eclipse.org/gef/>. Acesso em: 20 nov. 2011.

22. ECLIPSE FOUNDATION. GMF Dashboard View. Disponivel em: <http://wiki.eclipse.org/index.php/GMF_Tutorial_Part_4#GMF_Dashboard_View>. Acesso em: 01 dez. 2011.

23. ECLIPSE FOUNDATION. Graphical Modeling Framework Tutorial - Eclipsepedia. Disponivel em: <http://wiki.eclipse.org/Graphical_Modeling_Framework/Tutorial/Part_1>. Acesso em: 01 dez. 2011.

24. IBM. IBM Software - Rational Rose. Disponivel em: <http://www-01.ibm.com/software/awdtools/developer/rose/>. Acesso em: 01 dez. 2011.

25. OBJECT MANAGEMENT GROUP. Object Management Group - UML. Disponivel em: <http://www.uml.org/>. Acesso em: 01 dez. 2011.

26. KOLOVOS, D.; ROSE, L.; PAIGE, R. **The Epsilon Book**. [S.l.]: [s.n.], 2011.

27. ECLIPSE FOUNDATION. EuGENia. Disponivel em: <http://eclipse.org/gmt/epsilon/doc/eugenia/>. Acesso em: 21 nov. 2011.

28. ECLIPSE FOUNDATION. Emfatic Language Reference. Disponivel em: <http://eclipse.org/gmt/epsilon/doc/articles/emfatic/>. Acesso em: 2011 nov. 2011.

29. ECLIPSE FOUNDATION. Emfatic - Eclipsepedia. Disponivel em: <http://wiki.eclipse.org/Emfatic>. Acesso em: 20 nov. 2011.

30. ECLIPSE FOUNDATION. Epsilon Object Language. Disponivel em:

<http://eclipse.org/gmt/epsilon/doc/eol/>. Acesso em: 22 nov. 2011.

31. ECLIPSE FOUNDATION. Epsilon Validation Language. Disponivel em: <http://www.eclipse.org/gmt/epsilon/doc/evl/>. Acesso em: 01 dez. 2011.

32. ECLIPSE FOUNDATION. Epsilon Generation Language. Disponivel em: <http://www.eclipse.org/gmt/epsilon/doc/egl/>. Acesso em: 01 dez. 2011.

33. ECLIPSE FOUNDATION. Epsilon Wizard Language. Disponivel em: <http://eclipse.org/gmt/epsilon/doc/ewl/>. Acesso em: 22 nov. 2011.

34. W3SCHOOLS.COM. Javascript Introduction. Disponivel em: <http://www.w3schools.com/js/js_intro.asp>. Acesso em: 01 dez. 2011.

35. DEMUTH, B. OCL Portal - Technische Universität Dresden. Disponivel em: <http://www-st.inf.tu-dresden.de/ocl/>. Acesso em: 01 dez. 2011.

36. ORACLE CORPORATION. Oracle Database PL/SQL User's Guide and Reference 10g Release 2. **Oracle Docs**, jun. 2005. Disponivel em: <http://docs.oracle.com/cd/B19306_01/appdev.102/b14261.pdf>. Acesso em: 1 dez. 2011.

37. TECH ON THE NET. Oracle/PLSQL Topics. Disponivel em: <http://www.techonthenet.com/oracle/>. Acesso em: 1 dez. 2011.

38. ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados**. 6º. ed. São Paulo: Pearson Addison Wesley, 2011.

39. ECLIPSE FOUNDATION. Store Procedure (BIRT) - Eclipsepedia. Disponivel em: <http://wiki.eclipse.org/StoredProcedure_%28BIRT%29>. Acesso em: 05 dez. 2011.

40. ABRANTES, J. **Fazer monografia é moleza:** o passo a passo de um trabalho científico. 3º. ed. Rio de Janeiro: Wak Editora, 2011.

APÊNDICE A – Passo a passo para a criação do diagrama

Para que o usuário possa iniciar o trabalho de criação de procedimentos utilizando a ferramenta, primeiramente é preciso criar um projeto genérico ou qualquer outro tipo de projeto no Eclipse. Basta clicar com o botão direito do mouse na área do *Project Explorer* e escolher a opção *New – Project* como ilustrado na figura abaixo.

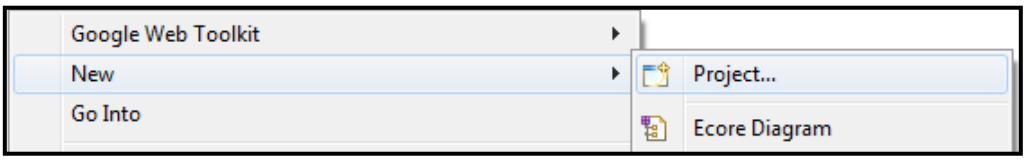

Figura Apêndice A.1 - Criação de novo projeto no Eclipse.

Uma janela será aberta mostrando várias opções. Selecionar a opção *General – Project* e clicar em *Next*. Ver a janela na ilustração abaixo.

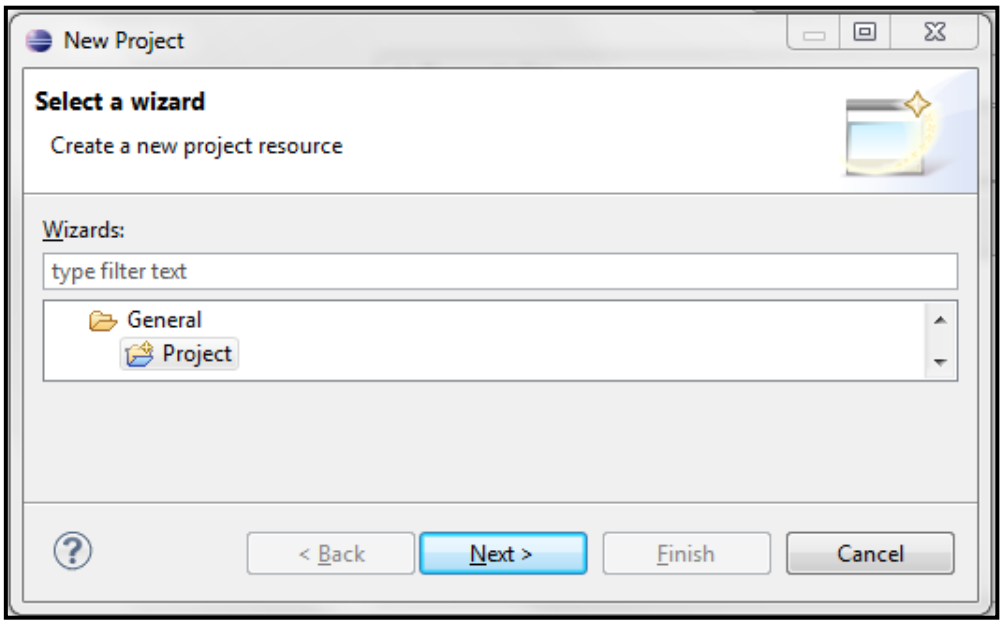

Figura Apêndice A 2 - Escolha do tipo do novo projeto.

Na próxima janela exibida, o nome do projeto deve ser configurado. Após isso, clicar em *Finish*. O projeto finalmente é criado e pode ser visualizado na aba Project Explorer.

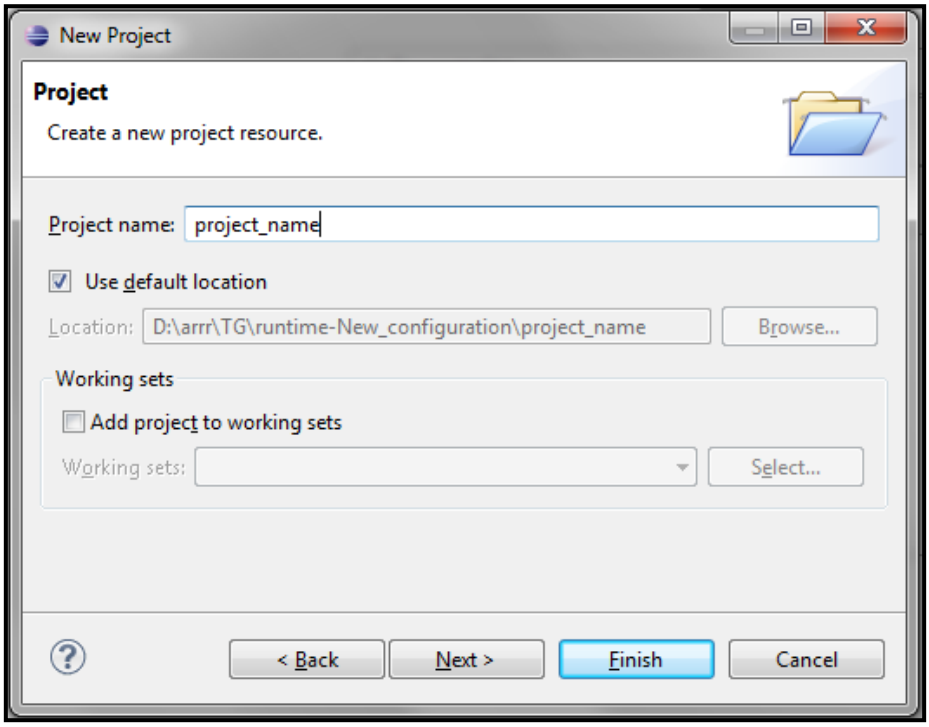

Figura Apêndice A 3 - Configuração do novo projeto.

Para criar um novo diagrama de Procedimento, clicar com o botão direito em qualquer projeto existente na aba *Project Explorer* e escolher a opção *New – Other* como ilustrado na figura abaixo.

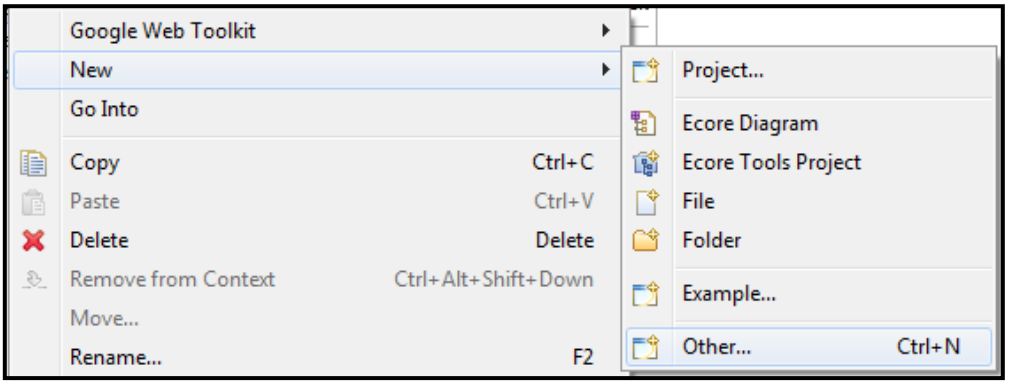

Figura Apêndice A 4 - Criação de novo arquivo.

Na janela que será aberta, escolher a opção *Examples – Procedure Diagram* e clicar em *Next*.

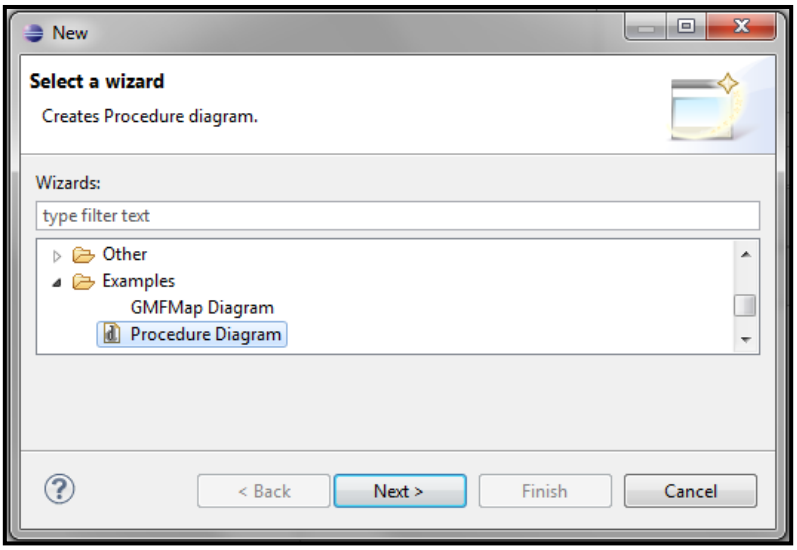

Figura Apêndice A 5 - Criação de novo diagrama de procedimento.

Na próxima janela, configurar o nome do arquivo do modelo de diagrama e clicar em *Next* novamente. Outra janela será aberta e o nome do arquivo do modelo de domínio deve ser configurado também. Após isso, clicar em *Finish*.

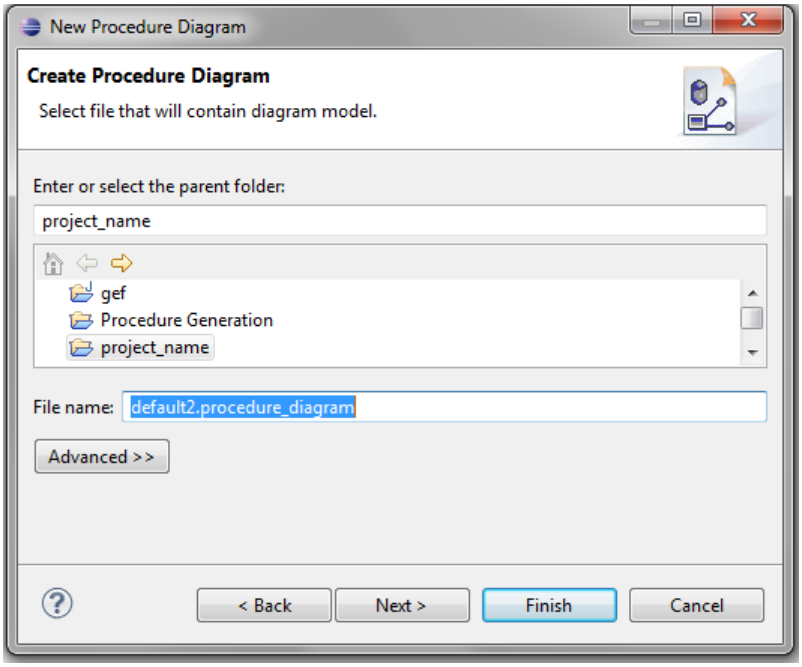

Figura Apêndice A 6 - Configuração dos nomes dos arquivos do diagrama.

Dois arquivos são criados e podem ser visualizados na aba *Project Explorer* dentro do projeto selecionado conforme mostrado na figura abaixo. Para abrir o diagrama, clicar duas vezes no arquivo com extensão *.procedure_diagram*.

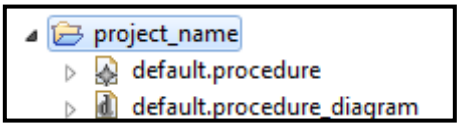

Figura Apêndice A 7 - Projeto e arquivos criados na aba *Project Explorer*.

ANEXO A – Exemplo de código *Emfatic*

```
@namespace(uri="filesystem", prefix="filesystem")
@gmf(foo="bar")
package filesystem;

@gmf.diagram(foo="bar")
class Filesystem {
    val Drive[*] drives;
    val Sync[*] syncs;
}

class Drive extends Folder {

}

class Folder extends File {
    @gmf.compartment(foo="bar")
    val File[*] contents;
}

class Shortcut extends File {
    @gmf.link(target.decoration="arrow", style="dash")
    ref File target;
}

@gmf.link(source="source", target="target", style="dot", width="2")
class Sync {
    ref File source;
    ref File target;
}

@gmf.node(label = "name")
class File {
    attr String name;
}
```

ANEXO B – Tipos de dados predefinidos em Oracle