UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

AMBIENTE PARA DESENVOLVIMENTO E CONVERSÃO DE FUNCTIONS DOS BANCOS DE DADOS ORACLE E SQL SERVER

RAFAEL MATIELLO

BLUMENAU
2007

2007/2-28

RAFAEL MATIELLO

AMBIENTE PARA DESENVOLVIMENTO E CONVERSÃO DE FUNCTIONS DOS BANCOS DE DADOS ORACLE E SQL SERVER

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciências da Computação — Bacharelado.

Prof. Alexander Roberto Valdameri, Mestre - Orientador

BLUMENAU
2007

2007/2-28

AMBIENTE PARA DESENVOLVIMENTO E CONVERSÃO DE FUNCTIONS DOS BANCO DE DADOS ORACLE E SQL SERVER

Por

RAFAEL MATIELLO

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Alexander Roberto Valdameri, Mestre – Orientador, FURB
Membro:	Prof. Maurício Capobianco Lopes, Mestre – FURB
Membro:	Prof. Wilson Pedro Carli, Mestre – FURB

Blumenau, 21 de novembro de 2007

Dedico este trabalho a minha família, pai, mãe, meus irmãos e namorada, que sempre me apoiaram e deram condições em todos estes anos de faculdade, a minha namorada, pela compreensão, paciência e por estar sempre comigo em todos os momentos, pelos amigos e colegas de trabalho, pelos auxílios e incentivos e ao meu orientador, por acreditar em min na realização deste trabalho.

AGRADECIMENTOS

À minha família, pai, mãe, Ricardo e Junior que mesmo longe, sempre estivaram presente me apoiando e aconselhando em todos os momentos.

A minha namorada, Suelen, que sempre amável esteve junto comigo, me apoiando, incentivando, motivando e fazendo com que eu enfrentasse os desafios e não desistisse de forma alguma.

Aos meus amigos e colegas de trabalho, pelos incentivos e ajudas prestadas durante o curso.

Aos professores do curso Ciências da Computação, pelos exemplos e conhecimentos repassados que contribuíram para a formação.

Ao meu orientador e amigo, Alexander Roberto Valdameri, por ter acreditado na minha capacidade e sempre que necessário auxiliado na conclusão deste trabalho.

Sonho que se sonha só é só um sonho que se sonha só, mas sonho que se sonha junto é realidade.

Prelúdio, Raul Seixas

RESUMO

Este trabalho apresenta um ambiente de desenvolvimento para criação de *functions* e *stored procedures* de banco de dados a partir de uma linguagem padrão. O ambiente efetua a geração de código baseada em *template*, utilizando o motor de *template* Velocity. O ambiente possui funcionalidades que auxiliam o desenvolvimento como sintaxe *highlight* e auto complemento de código com nomes de tabelas, colunas de bancos e sintaxe da linguagem.

Palavras-chave: Linguagem procedural. Geradores de código. Banco de dados. *Templates*. Ambiente de desenvolvimento.

ABSTRACT

This monography presents a development environment for creating functions and stored procedures of the database from a language standard. The environment makes the code generation based on template using the Velocity Template Engine. The environment has functionalities that help the development as highlight syntax and code completion with tables, database columns and syntax of language.

Key-words: Procedural language. Code generators. Database. Templates. Development environment

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

API – *Application Programming Interface*

AST - *Abstract Syntax Tree*

CSS - *Cascading Style Sheets*

CVS - *Concurrent Version System*

ERP - *Enterprise Resource Planning*

JDBC - *Java Database Connectivity*

JSP - *JavaServer Pages*

IDE - *Integrated Development Environment*

IDL - *Interactive Data Language*

LP - Linguagens Procedurais

PL/pgSQL - Procedural Language / PostgreSQL

PL/SQL - Procedural Language/ Structured Query Language

MVC - *Model-View-Controller*

RAD - *Rapid Application Development*

RMI - *Remote Method Invocation*

RPC - *Remote Procedure Call*

SGBD - Sistemas Gerenciadores de Bancos de Dados

SQL - *Structured Query Language*

SVN – Subversion

T-SQL – *Transact - Structured Query Language*

VTL - *Velocity Template Language*

UML – *Unified Modeling Language*

XML - *eXtensible Markup Language*

LISTA DE SÍMBOLOS

@ - arroba

$ - cifrão

% - por cento

SUMÁRIO

1 INTRODUÇÃO

Os Sistemas Gerenciadores de Bancos de Dados (SGBD) foram desenvolvidos para disponibilizar acesso facilitado aos dados, fornecendo funcionalidades como segurança, integridade, agilidade e confiabilidade no armazenamento das informações. Deitel e Deitel (2005, p. 895) definem que um sistema gerenciador de arquivos de banco de dados deve fornecer mecanismos para armazenar, organizar, recuperar e modificar os dados de muitos usuários, sem envolver a representação interna dos dados.

Os SGBD são classificados de acordo com seus modelos, onde os principais são o relacional, o orientado a objetos, os de redes e os modelos hierárquicos. Segundo Heuser (2000, p. 5), existem vários tipos de SGBD no mercado, sendo que o modelo relacional é o mais utilizado.

Confome Deitel e Deitel (2005, p. 895), bancos de dados relacionais utilizam a linguagem *Structured Query Language* (SQL), a qual é um padrão internacional, utilizada quase universalmente para efetuar consultas, solicitar informações a partir de determinados critérios e manipular dados.

Para escrever objetos de banco de dados existem linguagens procedurais que permitem desenvolver código de forma procedural, que nada mais são que conjuntos de códigos executados sequencialmente na SQL, combinando o poder de consulta do SQL com o das linguagens de auto nível, como C, Pascal e Java. As Linguagens Procedurais (LP) são utilizadas basicamente para criação de *triggers* (gatilhos), *functions* (funções) e *stored procedures* (bloco de comandos, em SQL ou em LP, que executam determinadas tarefas).

Os SGBDs implementam suas LPs com sintaxe própria e as denominam dependendo do fabricante. Por exemplo, no Oracle a linguagem utilizada é a Procedural Language/ Structured Query Language (PL/SQL), no Sql Server é utilizada a Transact- Structured Query Language (T-SQL) e no Postgres é utilizada a Procedural Language / PostgreSQL (PL/pgSQL).

A falta de padronização na sintaxe das LPs gera aos programadores de sistemas, que em geral utilizam mais de um SGBD, a árdua e complexa tarefa de conhecer detalhes de várias LPs. Como conseqüência, faz-se necessário um tempo e custo adicional, visto que, após a lógica descrita e validada em um SGBD, faz-se necessário converter para a sintaxe de outro gerenciador de banco de dados.

Tendo em vista a grande diversidade de opções de softwares que executam uma

mesma tarefa, como os sistemas de *Enterprise Resource Planning* (ERP), de gerência acadêmica, controle de estoque, contabilidade, entre outros e também clientes de grande, médio e pequeno porte, é essencial que as empresas de softwares oportunizem aos clientes, a opção de escolherem o banco de dados que melhor se ajuste às necessidades e infra-estrutura. Alguns pontos levados em consideração são: a massa de dados, quantidade de usuários, portabilidade em relação ao sistema operacional e os custos da aquisição de um SGBD.

Considerando a necessidade das aplicações serem compatíveis com vários SGBDs e das LPs não fazerem uso de uma sintaxe padrão, foi implementado um software que contemple um ambiente de desenvolvimento de *functions* de banco de dados.

O ambiente implementado auxilia no desenvolvimento e na criação de *functions*, pois a partir de uma *function* desenvolvida na LP padrão, é possível transcrever para outras LPs que se adequarem aos *templates*, que são utilizados para efetuar a conversão entre a LP padrão do ambiente e as LP customizadas em forma de *templates*, possibilitando diminuir o "retrabalho", eliminar a necessidade de conhecer todas as sintaxes das LPs, agilizando o processo de desenvolvimento, assim sendo possível diminuir o tempo e os custos de desenvolvimento.

1.1 OBJETIVOS DO TRABALHO

O objetivo geral deste trabalho é desenvolver um ambiente de desenvolvimento de *functions* de banco de dados a partir de uma linguagem procedural padrão, com a funcionalidade de gerar *functions* para os padrões Oracle (PL/SQL) e Sql Server (T-SQL) utilizando-se de *templates* para a conversão entre as LP.

Os objetivos específicos do trabalho são:

a) definir uma linguagem padrão para criação de function de SQL, contemplando os recursos de uma linguagem procedural;
b) disponibilizar um ambiente de execução da linguagem desenvolvida, contendo um compilador que fará a verificação léxica, sintática e semântica para validação dos códigos escritos em linguagem procedural;
c) conectar aos bancos de dados, inserir as functions e obter informações dos SGBDs;
d) incluir no editor, funcionalidades como auto completar de comandos da linguagem, sugerindo nomes das tabelas e colunas do SGBD e sintaxe *highlight*;

e) gerar functions para as linguagens PL/SQL e T-SQL;
f) definir templates de configuração de sintaxe para SGBDs, possibilitando adicionar novos SGBDs para conversão.

1.2 ESTRUTURA DO TRABALHO

O texto está estruturado em quatro capítulos. No segundo capítulo é conceituada a fundamentação teórica do trabalho que consiste em apresentar um ambiente de desenvolvimento descrevendo suas características, vantagens e desvantagens e dois exemplos, os quais são o NetBeans e o PL/SQL Developer. Após, é conceituado LP e apresentado um comparativo entre duas LPs a PL/SQL e a T-SQL. O capítulo ainda apresenta a geração de código, *templates* e motor de *templates* Velocity e trabalhos correlatos.

No terceiro capítulo é apresentado o desenvolvimento do ambiente, incluído especificação de requisitos, digramas de casos de uso, atividades e seqüência, as ferramentas utilizadas no desenvolvimento, a implementação e a operacionalidade do sistema.

No último capítulo são apresentadas as conclusões e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados: ambiente de desenvolvimento, linguagem procedural em SGBD, diferenças entre as LP PL/SQL da Oracle e T-SQL do Sql Server, geração de código, *templates* e motor de *template* Velocity e trabalhos correlatos.

2.1 AMBIENTE DE DESENVOLVIMENTO

Os ambientes de desenvolvimento de softwares ou *Integrated Development Environment* (IDE) são ambientes de desenvolvimento integrados, que auxiliam no desenvolvimento de sistemas ou parte deles durante o processo de implementação. As IDEs contemplam aspectos de potencialidade, funcionalidades e produtividade (SEVERO, 2005, p. 1-2).

As IDEs tem por finalidade auxiliar no desenvolvimento de software, através da disponibilidade de um conjunto de funcionalidades que permitem o controle do processo de desenvolvimento da aplicação, controle que envolve os programadores alocados no projeto e o conjunto de ferramentas de software envolvido no processo.

Algumas vezes as IDEs são confundidas e tituladas utilizando o termo *Rapid Application Development* (RAD), que é desenvolvimento rápido de aplicações. Normalmente existe esta confusão em relação ao aumento na velocidade de produção. O aspecto RAD de uma IDE está no fator produtividade, pois a produção é apoiada por um conjunto de ferramentas que otimizam o processo. Assim, a implementação do projeto é acelerada, aumentando a produtividade da equipe (SEVERO, 2005, p. 2).

O uso de ambientes integrados costuma aumentar a produtividade dos desenvolvedores, principalmente na construção das interfaces de um software onde as janelas são padronizadas e constantes nos projetos.

2.1.1 CARACTERÍSTICAS

Segundo Severo (2005, p. 3-5), a principal característica de uma IDE é a concentração

em um único ambiente das funcionalidades que contemplem todas as etapas da construção de um software e possua as ferramentas necessárias para o projeto, tais como:

a) editor de código fonte: conter um editor integrado possibilitando ao desenvolvedor a elaboração de código fonte. As principais funcionalidades de um editor são: busca e substituição, marcadores, destaques de cores em palavras-chaves, teclas de atalho, ferramenta para auto completar instruções, ocultação de instruções de bloco, ajuda entre outras;
b) compilador: a partir de um comando, o sistema compila os códigos fontes e exibe os resultados, em caso de erros, o sistema localiza o erro, normalmente é feito destacando o posicionamento do cursor ou destacando a linha que contenha o erro e exibe uma mensagem para auxiliar na resolução;
c) execução: similar ao compilador, normalmente é ativado com um comando e executa o resultado obtido com a compilação dos fontes. Se não ocorrer erros, exibe os resultados;
d) depuração: os depuradores ou *debugger* são utilizadas para verificação de código fonte durante o processo de execução. As verificações normalmente são feitas em busca de erro de lógica;
e) ajuda: com a evolução dos sistemas, os comentários inseridos no próprio código fonte atuam como auxiliares para identificar funções;
f) auxiliares: são os chamados *wizards*, que são módulos embutidos nos sistemas para auxiliar no desenvolvimento a partir perguntas feitas ao desenvolvedor;
g) gerenciadores de projetos: toda aplicação de software deve ser encarada com um projeto composto por conjuntos de módulos, documentação, informações sobre execução, controle de versão, entre outros. Para controlar estes processos é utilizado um gerenciador;
h) janela de execução: esta janela deve apresentar as informações, mensagem, erros que poderão ocorrer durante a execução;
i) customização do ambiente: todo ambiente deve ser possível customizar conforme as necessidades específicas do programador, desde o aspecto de cores do editor, teclas de atalho, botões, entre outros e se possível ter abertura para expansão por meio de *plug-ins*;
j) sistema de controle de versão: é um recurso essencial nos projetos executados através de equipes. As implementações são distribuídas entre os desenvolvedores e a controle de versão dos códigos fontes e consolidação do mesmo fica sob

responsabilidade do controle de versão.

2.1.2 VANTAGENS E DESVANTAGENS

A utilização de ambiente de desenvolvimento de software gera muita discussão entre as comunidades de desenvolvedores. Conforme cita Severo (2005, p. 6), existem críticas a respeito da geração de código automaticamente por parte das IDEs, onde o código gerado muitas vezes não é o ideal e possuem muitas linhas de código a mais do que o necessário. Outra desvantagem citada é que as IDEs limitam a linguagem a seu ambiente. Em alguns ambientes, o desenvolvedor se transforma em construtor de interfaces e responsável por ligar componentes.

Por outro lado, as IDEs são de muita utilidade para desenvolvedores, que estão iniciando os estudos da linguagem e ainda não possuem domínio sobre a mesma. Severo (2005, p. 6), explica que as limitações do ambiente podem servir de desafio aos desenvolvedores que tiverem interesse em aperfeiçoar o conhecimento sobre a tecnologia.

Um ponto forte e favorável da utilização de IDEs é a produtividade, que muitas vezes determina a viabilidade da utilização da IDE em um projeto. A grande produtividade vem da não necessidade de escrever tanto código, como é o exemplo no caso de interfaces padronizadas, onde as ferramentas normalmente fazem boa parte da codificação.

Segundo Severo (2005, p. 6), sempre existirão limitações quanto às IDEs, mas com a disposição de novos componentes e funcionalidades, a utilização de IDEs sempre será recomendada.

As IDEs são divididas basicamente em dois grupos: IDEs proprietárias e livres. IDEs proprietárias são as comercializas sobre uma licença de uso impostas pelas empresas produtoras. Algumas IDEs deste grupo são: Delphi da Borland, PL/SQL Developer da Alround Automations, WebSphere da IBM. As IDEs livres, são, como o próprio nome já diz, de uso livre. Entre as mais populares podem ser citadas o NetBeans IDE, SQL Developer da Oracle e o projeto Eclipse, que é mantido pela IBM.

A seguir são apresentados dois ambientes de desenvolvimentos e algumas funcionalidades.

2.1.3 NETBEANS

A NetBeans IDE originou-se de um projeto determinado Xelphi, em 1996, com objetivo de criar uma ferramenta de rápido desenvolvimento, semelhante ao Delphi, voltado à programação Java. O projeto despertou o interesse da Sun Microsistems, que adquiriu os direitos sobre o mesmo (SEVERO, 2005, p. 12).

O projeto NetBean é uma iniciativa mantida pela comunidade mundial de desenvolvedores, e não tem fins lucrativos e sim de aprendizado e aplicação do conhecimento.

A interface do ambiente é flexível e altamente customizável, organizadas em módulos e divididas em áreas de trabalho com a finalidade de melhor organização dos componentes e visões. As caixas de ferramentas podem ser posicionadas em qualquer lugar do layout (SEVERO, 2005, p. 9).

Os projetos são organizados de forma hierárquica, permitindo assim, fácil acesso à informação. Para acessar o conteúdo dos arquivos basta clicar sobre o arquivo e a IDE seleciona um editor para exibir o conteúdo (SEVERO, 2005, p. 10).

O editor possui várias funcionalidades para auxiliar no desenvolvimento, como, coloração de sintaxe para linguagens como Java, JSP, XML, CSS e IDL, auto completar, *bookmarks* e parser de código durante o desenvolvimento.

O grande diferencial do NetBean é a interface para desenvolvimento gráfico, que possui suporte a componentes visuais e não visuais, que pode ser customizável com componente das bibliotecas AWT, Swing e outras compatíveis.

O NetBeans disponibiliza facilidades como *wizards* para configurar ferramenta de controle de fonte e versão como CVS.

O depurador da IDE é bastante útil na verificação de códigos, possui funcionalidades como ponto de parada usando expressões, visualização de valores de variáveis e estruturas de dados. É possível depurar múltiplas sessões ao mesmo tempo e remotamente.

A IDE possui suporte a várias linguagens como C, C++, XML, HTML, entre outras, e a partir de uma de suas interfaces também é possível conectar a bancos de dados como Oracle, IBM DB2, MySQL, PostgreSQL.

O desenvolvimento de projetos web também é suportado pelo NetBeans através de JSP e Servlets, contendo *templates* para agilizar o processo de desenvolvimento. A ferramenta possui integração com o Tomcat para facilitar os testes e avaliação.

Além das funcionalidades citadas, possui suporte a outras tecnologias como Ant,

utilizado na construção de projetos, RMI, CORBA para aplicações distribuídas entre muitas outras funcionalidades.

A Figura 1 apresenta a interface da IDE NetBeans versão 5.5.1.

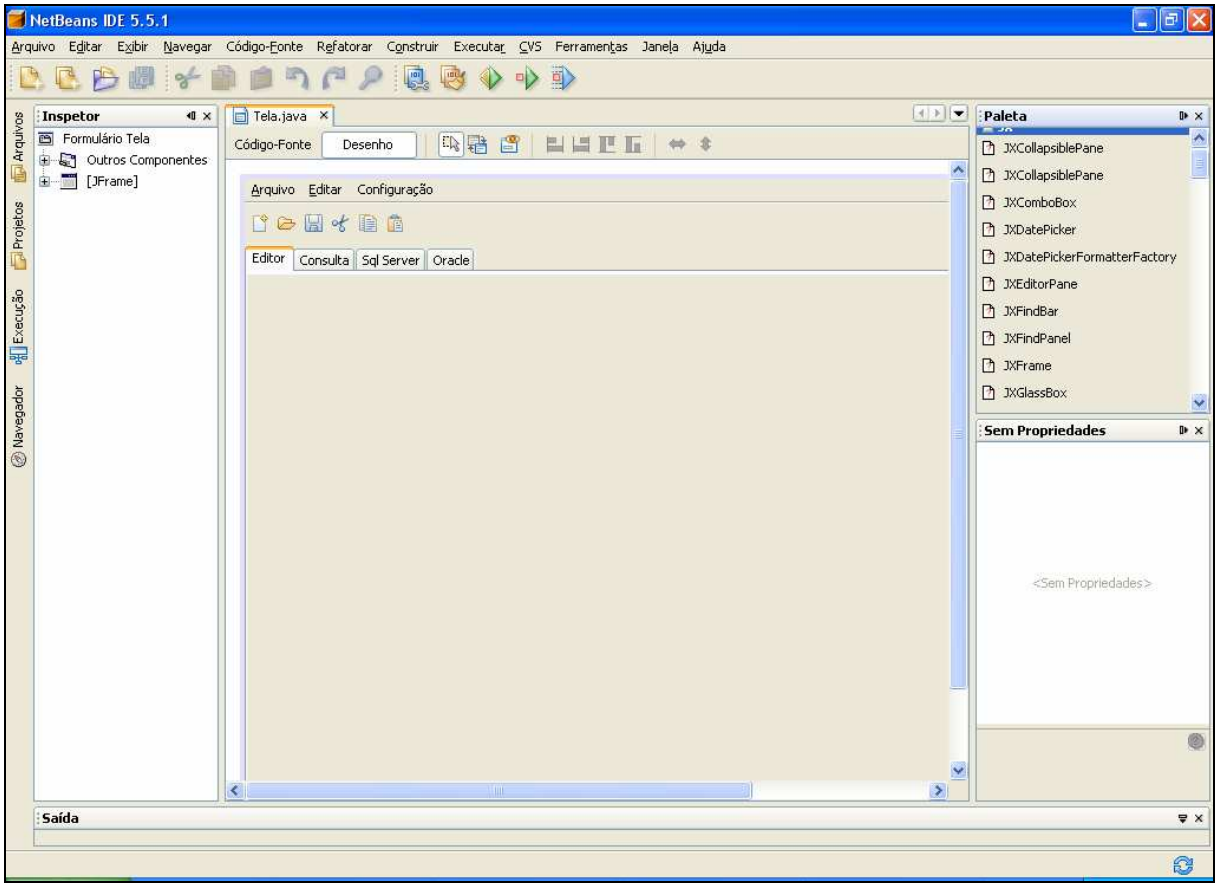


Figura 1 – NetBeans IDE 5.5.1

2.1.4 PL/SQL DEVELOPER

O PL/SQL Developer é uma IDE para desenvolvimento de LP, criada pela empresa Allround Automations para o desenvolvimento de LP (*functions*, *procedures*, *triggers*, *packages*) armazenados em um banco de dados Oracle.

O PL/SQL Developer oferece uma interface amigável e produtiva para as tarefas de edição, compilação, correção, testes, otimização e consulta de programas na aplicação cliente-servidor em Oracle, além de outras funcionalidades como execução de *scripts* SQL, criação e modificação de definições de tabelas (através de linguagem DDL) e relatórios (REZENDE, 2004).

Segundo Rezende (2002), o PL/SQL Developer permite a criação de perfis de usuários

com permissões de acesso, para otimização de código, possui um utilitário o *Explain Plan*, que permite verificar o trajeto da execução das expressões e depurador para verificação de possíveis erros.

Urman (2002, p. 59) destaca também características importantes como o suporte a sistema de terceiros para controle de versão (efetuado por meio de *plug-in* externo), navegador de objetos que permite a visualização de todos os objetos acessíveis do banco de dados, registro de macros, onde é possível gravar macros de rotinas que são executadas com freqüência e destaque de erros ocorridos na execução de LPs.

A Figura 2 apresenta a IDE PL/SQL Developer.

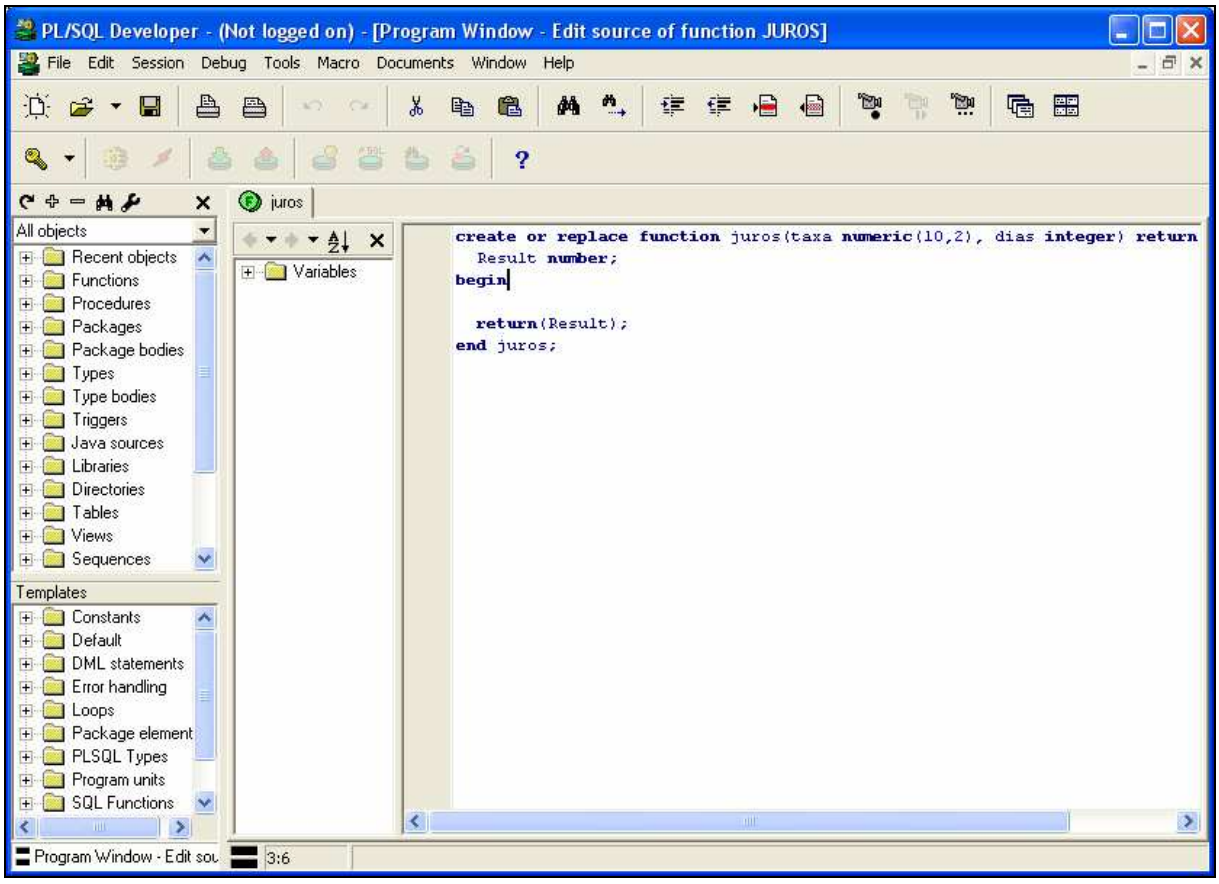


Figura 2 - PL/SQL Developer

2.2 LINGUAGENS PROCEDURAIS

Segundo O´Hearn (2002, p. 4), as LPs foram criadas a partir da combinação do SQL,

com a comodidade gerada pelas linguagens populares de terceira geração, entre elas C, FORTRAN, COBOL, Ada, Pascal e PL/1. LPs possuem características de declarações de variáveis, lógicas condicionais, laços de repetições, subprogramas, *parser* de parâmetros, controle de erro, sendo que todas estas funcionalidades são integradas com a linguagem SQL.

Os programas desenvolvidos em LP são programas unitários com estruturas próprias, como as *stored procedures* e as *functions*. Programas unitários podem estar localizados junto ao servidor ou ao cliente, possibilitando assim boa flexibilidade e um melhor desempenho (O´HEARN, 2002, p. 4).

Algumas vantagens para o uso da LP citadas por Fanderuff (2000, p.105) são:

a) portabilidade: qualquer máquina que interaja com o banco de dados pode executar a mesma aplicação;
b) integração com o gerenciador de banco: uma vez que as variáveis são definidas a partir de colunas no banco de dados e as alterações podem ser realizadas automaticamente no próprio banco de dados, ou seja, sem manutenção, isso pode refletir nos blocos da LP;
c) capacidade procedural: comandos de repetição, controle de fluxo e tratamento de erros;
d) produtividade: desenvolvimento de *stored procedures*, *functions* e *trigger* de banco de dados.

Segundo Urman (2002, p. 4), o SQL, sendo considerada uma linguagem de quarta geração, é relativamente simples se comparada às de terceira geração e com a facilidade de possuir menos comandos. Urman (2002, p. 5-6) afirma que pelo fato das linguagens procedurais serem executadas no banco de dados, o tráfego de rede em aplicações cliente servidor diminui consideravelmente, pois varias instruções SQL podem ser empacotadas em bloco de LP e enviadas ao servidor como uma única unidade. Isto resulta em menos tráfegos de rede e uma aplicativo mais rápido. Até mesmo quando aplicativos cliente servidor são executados no servidor, o desempenho aumenta. Neste caso, a comunicação com o SGBD, mesmo sem a necessidade da rede, resulta em menos acesso ao banco de dados.

Fanderuff (2000, p. 153) explica que *functions* são subprogramas que têm por objetivo retornar algum resultado ou valor. As funções de um banco de dados podem ser utilizadas nas aplicações como qualquer função SQL, ou seja, em atribuição a variáveis ou como argumentos em comandos `SELECT`.

2.3 PL/SQL VERSUS T-SQL

Os SGBDs, que contemplam linguagens procedurais, definem sintaxe de linguagens próprias, que são semelhantes em muitos aspectos, mas não idênticas, conforme Postgresssql (2007). Esta seção tem a função de fazer uma comparação entre PL/SQL e T-SQL, utilizadas no desenvolvimento de *functions.* O Quadro 1 mostra alguns exemplos de diferenças de sintaxe entre as linguagens.

	PL/SQL	T-SQL

Criação de functions	create or replace
	function First5
	(p_Char varchar2)
	return varchar2(5)
	is
	begin
	return
	substr(p_Char, 1, 5);
	end;
		create
		function First5
		(@p_Char varchar(20))
		returns varchar(5)
		as
		begin
		return substring(@p_Char, 1
		, 5);
		end

Declaração de variáveis	declare v_job varchar2(10);	declare @job_desc varchar(50)
Atribuição de variáveis	pi := 3.14159;	set @pi = 3.14159;

Condições	if 0 > 1 then
	saida := '0 > 1';
	elseif 0 > 2 then
	saida := '0 > 2';
	else
	saida := 'None';
	end if;
		if 0 > 1
		set @saida = '0 > 1'
		else
		if 0 > 2
		set @saida = '0 > 2'
		else
		set @saida = 'None'

Declaração de cursos	declare Nome_do_cursor cursor fast_for_forward for select al_nomalu, al_codalu from tb_aluno where al_codalu = @codalu	CURSOR Nome_doCursor IS select al_nomalu, al_codalu from tb_aluno where al_codalu = :pCodalu
Loops	open Nome_do_cursor; fetch Nome_do_cursos into @nomalu, @codalu while @@fetch_status = 0 begin … end	OPEN Nome_do_cursor FETCH Nome INTO al_nomalu, al_codalu; WHILE nome%FOUND LOOP begin end loop;
Converter Numérico para Caracter	to_char(expressão,formato)	str(expressão,tamanho,decimal)
Converter caractere para Numérico	to_number(expressão,formato)	Cast(expressão AS Numeric)
Converter caractere para Data	to_date(expressão,formato)	Cast(expressão AS DateTime)
Retornar parte de uma expressão	SubString(expressão,início,tamanho)	SubStr(expressão,início,tamanho)
Concatenar expressões	+	||
Testa se o valor do campo é nulo, caso for retorna a expressão senão o campo	nvl(Campo, Expressão)	isnull(Campo, Expressão)
Buscar uma string em outra string	INSTR	CHARINDEX
Tamanho da String	LENGTH	DATALENGTH

Quadro 1 - Comparativo PL/SQL da Oracle e T-SQL do Sql Server

2.4 GERAÇÃO DE CÓDIGO

Geração de código é uma técnica que usa programas para gerar programas. Geração de código é apenas uma das etapas de um compilador. As etapas de um compilador são dividas em duas partes análise e geração de código. A etapa de analise compreende o analisador

léxico, sintático e semântico e a geração de código é composta por gerador de código intermediário, otimizador de código e geração de código final (PRICE e TOSCANI ,2001, p. 4-5). Geradores podem ser desde simples formatadores de código até ferramentas que geram aplicações complexas a partir de modelos abstratos (HERRINGTON, 2003).

Segundo Herrington (2003, p. 15), a geração de código fornece benefícios à Engenharia de Software em vários níveis. Alguns desses benefícios são:

a) qualidade: os códigos desenvolvidos pelos programadores tendem a ter qualidade uniforme. Já códigos gerados a partir de *templates*, são códigos com qualidade padrão, consistentes e livres de falhas;
b) consistência: o código que é construído por um gerador tende a ser consistente e mais fácil de ser compreendido e usado. Além disso, para incluir novas funcionalidades ou corrigir erros, é necessário alterar os modelos e executar o gerador, sendo que as alterações são aplicadas consistentemente em todo o código;
c) único ponto de conhecimento: caso o gerador tenha o objetivo de gerar código para diferentes camadas da aplicação, basta o desenvolvedor alterar o gerador em apenas um ponto, para replicar a alteração para os outros processos seguintes;
d) maior tempo para o projeto: o cronograma de um projeto que utiliza a geração de código é completamente diferente do cronograma de um projeto onde o desenvolvimento é totalmente manual. Em um projeto que adota o desenvolvimento manual, o uso inadequado de uma API pode ocasionar a reescrita de uma grande quantidade de código. Com o uso de geradores de código, os desenvolvedores podem modificar os *templates* para adequar-se a API e executar o gerador novamente, e todo o código gerado estará modificado contemplando as necessidades;
e) decisões atípicas do projeto: o alto nível das regras de negócio é perdido em minuciosas implementações de código. Os geradores de código utilizam arquivos com abstrações de definições dos códigos que serão gerados estes arquivos que normalmente são muito pequenos, com poucas linhas de código e possuem a especificação da geração de código, tornando assim muito mais fácil uma correção, considerando que possui algumas linhas de código e não centenas para serem analisadas;
f) abstração: gerar código a partir da definição da lógica da aplicação, usando um modelo abstrato, permite que o modelo especificado seja revisto e validado mais facilmente, já que é menor e mais específico do que o código resultante. Ainda, o

gerador pode ser reprojetado para traduzir a aplicação para outras linguagens, tecnologias ou plataformas;

g) produtividade: o cronograma de desenvolvimento de um projeto usando geração de código é significativamente diferente de um projeto codificado manualmente.

Para Herrington (2003, p. 77), a construção de um gerador de código pode seguir as seguintes etapas de desenvolvimento:

a) identificação do que se deseja obter como saída do gerador;
b) definição de qual será a entrada e analisadores;
c) interpretação e recuperação das informações da entrada, definindo a formatação e a geração da saída;
d) geração da saída a partir das informações extraídas do arquivo de entrada.

Herrington (2003, p. 25) indica que a geração de código é genuinamente útil quando é usada para criar um volume razoável de dados, quando as características do que é pretendido converter forem estáveis, como por exemplo, acesso a banco de dados, *stored procedure* e *Remote Procedure Call* (RPC).

A geração do código é uma ferramenta extremamente valiosa, que pode ter impacto na produtividade e na qualidade de projetos. Compreender o funcionamento das ferramentas usadas no processo de desenvolvimento de software pode facilitar o uso de técnicas de geração do código (HERRINGTON, 2003, p. 27).

2.5 TEMPLATES

Takecian (2004, p. 6) define que *templates* são arquivos que descrevem uma estrutura de formatação, contendo diretivas indicando valores a serem inseridos e ações a serem executadas pelo sistema. *Templates* possuem o objetivo de gerar outros arquivos seguindo o modelo ou *template* definido, onde é necessário alterar somente o conteúdo, sendo assim indicado para geração de código.

Um *template* possui códigos estáticos e dinâmicos. Códigos dinâmicos são aqueles que passarão pelo processo de transformações de código gerando o código de saída. Código estático e a parte do código que não será modificada pelo motor de *templates* (KLUG, 2007, p. 24).

Os *templates* são bastante utilizados em áreas do sistema, onde as informações

precisam ser dinâmicas e podem ser alteradas por usuários através de variáveis, sem a necessidade de novas implementações.

2.6 MOTOR DE TEMPLATES *VELOCITY*

Motores de *templates* são mecanismos que viabilizam a separação do código dinâmico do código estático, possibilitando o desenvolvimento independente entre as partes. O *Velocity* é um motor de *templates* desenvolvido na linguagem Java. Permite usar uma línguagem simples e poderosa de *templates* para referenciar os objetos definidos no código de Java. Velocity é muito utilizado no desenvolvimento de páginas web, mas pode ser utilizado para gerar SQL, PostScript, XML, na geração de relatórios. Pode ser utilizada sozinha ou integrada com outros sistemas (APACHE SOFTWARE FOUNDATION, 2004).

Moura e Cruz (2002), na concepção inicial o Velocity enfatiza a organização dos sistemas interativos utilizando a arquitetura MVC. A atuação básica do Velocity consiste na a formatação e apresentação dos dados, ou seja, na camada de visão dos sistemas.

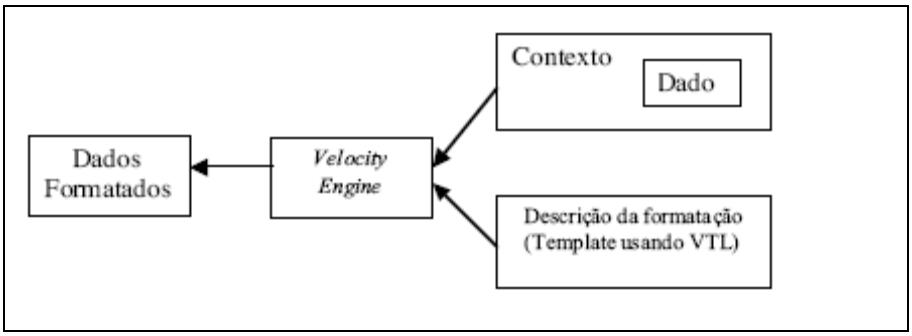
O Velocity utiliza-se de uma linguagem denominada *Velocity Template Language* (VTL). A VTL é uma linguagem simples, contando com poucos comandos e que permite a definição de variáveis, controle de fluxo de execução, inclusão de *templates* e definição de macros. A simplicidade da linguagem facilita a aprendizagem aos *designers* de páginas e usuário (MOURA; CRUZ, 2002).

A Figura 3 apresenta a arquitetura da ferramenta que é dividida em quatro partes: contexto, descrição de formatação, motor Velocity e os dados formatados, conforme descrito por Moura e Cruz (2002), são elas:

a) descrição da formatação: são os *templates*, que devem ser feitos a partir da linguagem de *templates* VTL;
b) contexto: é implementado pela classe org.apache.velocity. VelocityContext do pacote da Velocity, e a sua função é nomear os objetos da aplicação para que eles possam ser referenciados no *template*. Os nomes são apresentados em forma de *string* Java e são utilizados pelo motor de *templates* para encontrar as referencias e inserir os objetos;
c) motor Velocity: responsável por interpretar o *template* e nesta interpretação encontrar os mapeamentos feitos no contexto, para identificar e acessar o banco de

dados de objetos;

d) dados formatados: é o resultado da interpretação do *templates* pelo motor do Velocity.



Fonte: Moura e Cruz (2002).

Figura 3 – Arquitetura do motor de *templates* Velocity

Durante uma execução normal do Velocity, são realizados os seguintes passos (MOURA; CRUZ, 2002):

a) inicialização do Velocity, carregando uma configuração que definirá algumas características de seu funcionamento;
b) criação do contexto, mapeamento entre referências no arquivo de formatação em VTL e as classes Java é especificado;
c) o Velocity carrega *template* especificado;
d) o *template* é analisado e armazenado em memória na forma de uma estrutura de dados conhecida como AST;
e) a representação AST é utilizada para realizar a substituição adequada onde for necessário, das referências pelos valores obtidos dos objetos Java, de acordo com as regras de mapeamento definidas pelo contexto.

O Velocity pode ser utilizado para desenvolvimento utilizando padrões de projeto MVC, onde no desenvolvimento de um sistema, a etapa inicial seria identificar os layout dos *templates* e padrões de nomenclaturas, para que os desenvolvedores e *designers* trabalhem paralelamente, agilizando assim o processo de desenvolvimento, sem a necessidade dos designers se preocuparem com detalhes complexos e de funcionamento da linguagem e o desenvolvedores com o *layout*.

2.7 TRABALHOS CORRELATOS

Existem alguns trabalhos que possuem a mesma linha de pesquisa deste, cada um com seus diferenciais e particularidades, como linguagens utilizadas nas implementações, diferentes formas de geração e customização de código. Dentre os trabalhos pode-se citar Hiebert (2003) e Bianchi (2006) que construíram geradores de código para *stored procedured*, e Klug (2007) que implementou geração de páginas JSP no padrão MVC, a partir do metadados de banco de dados relacional utilizando o motor de *templates* Velocity.

Hiebert (2003) abordou a construção de um protótipo de compilador para a linguagem PL/SQL voltado a *stored procedures*, cujo objetivo é a geração de código para diversos SGBDs, onde foi definida uma gramática de uma linguagem padrão e implementandas funcionalidades como verificação de erros e geração de código para um SGBD. Bianchi (2006) efetuou extensão no trabalho implementando melhoramentos como detecção de erros através de análises léxica e sintática, conexão com os SGBDs Oracle, Sql Server e Postgres para fins de validação, adicionou funcionalidades para salvar e abrir arquivos desenvolvidos e incrementou a sintaxe da linguagem. Ambos foram desenvolvidos em Delphi e voltados a *stored procedures*.

Klug (2007) apresentou uma ferramenta para efetuar a geração de páginas JSP no padrão MVC, a partir do metadados de um banco de dados relacional. Através da API JDBC é extraído o dicionário de dados dos SGBDs Oracle, Microsoft SQL Server e MySQL. Utilizando as informações extraídas do metadados, a ferramenta permite ao desenvolvedor criar um projeto, configurar e salvar as definições de formulários, grupos e relatórios. As páginas JSP a serem geradas, utilizando as informações configuradas, são definidas através de *templates* e a geração é realizada utilizando o motor de *templates Velocity*.

3 DESENVOLVIMENTO DA FERRAMENTA

Este capítulo apresenta o processo de desenvolvimento do ambiente, descrevendo as ferramentas e técnicas utilizadas durante sua implementação, onde foram contemplados os seguintes passos:

a) identificação, análise e especificação dos requisitos funcionais e não funcionais do ambiente;
b) estudo das LPs dos SGBDs Oracle XE e Microsoft SQL Server 2005, analisando as estruturas, semelhanças e diferenças na criação de *functions*;
c) desenvolvimento de uma linguagem procedural para atender os estudos efetuados nos itens acima;
d) definição dos *templates* levando em considerações as diferenças entre as LPs;
e) especificado do ambiente com análise orientada a objetos utilizando a UML, para desenvolvimento dos diagramas de caso de uso, de classes, de atividade e de seqüência;
f) efetuado pesquisas sobre o funcionamento e implementações de funcionalidades de auto complemento e sintaxe *highlight*;
g) implementado as conexões com banco de dados utilizando a API JDBC e efetuado estudos sobre a obtenção do metadados para incluir as informações obtidas na lista de auto completar;
h) implementado geração de código utilizando a ferramenta GALS e o motor de *template* Velocity;
i) implementado consultas utilizando `SELECT`, fazendo uso das conexões desenvolvidas;
j) criado um banco de dados e funções que contemplem as funcionalidade das LPs para efetuar testes nos bancos de dados Oracle XE e Microsoft Sql Server.

3.1 LINGUAGEM PROCEDURAL

A linguagem do ambiente é baseada na definida por Valdameri (2007), a partir desta linguagem é foi removido comandos considerados desnecessário para criação de *functions* e

stored procedure como: criação, alteração e exclusão de tabelas, *constrains* e *index.* As funcionalidades adicionadas na linguagem foram:

a) comandos de criação de *functions* e *stored procedure*;
b) condição como `if` e `else`;
c) comandos de repetição como `while`, `for` e `loop`;
d) declaração de variáveis;
e) atribuição de valores a variáveis;
f) manipulação de cursores;
g) tratamento de exceções;
h) funções de string, data, numéricas e conversão;
i) incrementado opções do comando select adicionando comandos de `joins`, funções;
j) retorno de valores.

Mais informações sobre a linguagem e sua sintaxe podem ser consultadas no Apêndice A que contem a BNF da linguagem.

3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O levantamento de requisitos consiste na etapa de compreensão do problema aplicado ao desenvolvimento do software e tem como principal objetivo que, tanto desenvolvedor quanto usuários, tenham a mesma visão do problema a ser resolvido pelo sistema (BEZZERA, 2003, p. 20-21). No Quadro 2 são apresentados os requisitos não funcionais que declaram características que o sistema deve possuir e são relacionadas às funcionalidades e no Quadro 3 são apresentados os requisitos funcionais que definem a funcionalidade do sistema.

REQUISITOS NÃO FUNCIONAIS
RNF001 - Ser implementado utilizando ambiente de desenvolvimento NetBeans.
RNF002 - Utilizar linguagem Java para implementação.
RNF003 - Utilizar o software GALS para gerar o analisador léxico, sintático e semântico.
RNF004 - Utilizar driver JDBC para conectar os bancos de dados.
RNF005 – Utilizar *templates* para a geração do código de saída.
RNF006 – Utilizar o motor de *templates* Velocity para formatação dos *templates*.

Quadro 2 – Requisitos não funcionais

REQUISITOS FUNCIONAIS	CASO DE USO
RF001 - Criar um script com a LP.	UC008
RF002 - Implementar um editor para a escrita de código fonte da LP com funcionalidade de auto complemento de texto e sintaxe *highlight*.	UC008
RF003 - O ambiente deverá processar *functions* contendo declarações de variáveis, atribuições, testes condicionais, laços de repetições e converter para o padrão PL/SQL e T-SQL.	UC004
RF004 - Permitir inserir as *functions* convertidas no banco de dados.	UC005
RF005 - Conectar em banco de dados Oracle e Sql Server.	UC001
RF006 - Efetuar análise léxica e sintática para a LP.	UC006
RF007 - Permitir ao usuário salvar o código fonte em um arquivo.	UC008
RF008 - Possibilitar a abertura de um arquivo já salvo, para edição ou geração de código.	UC008
RF009 - Possibilitar a configuração para novos SGBDs utilizando *templates* em um formato padrão para serem customizados	UC002
RF010 – Permitir executar consultas aos bancos de dados.	UC003
RF011 – Permitir configurar conexões com bancos de dados utilizando JDBC.	UC001
RF012 – Salvar as informações de conexão com o banco de dados.	UC001
RF013 – Permitir testar as conexões com os bancos de dados.	UC001
RF014 – Permitir ler informações do metadados do banco de dados e incluir no auto completar o nome das tabelas e suas colunas no banco de dados.	UC007
RF016 – Permitir importar *templates* de banco de dados.	UC002
RF017 – Permitir exportar *templates* de banco de dados.	UC002
RF018 – Permitir clonar *templates* de banco de dados.	UC002
RF019 – Possibilitar ativar ou inativar um *template* quanto sua utilização, sem a necessidade de excluir o *template*.	UC002

Quadro 3 – Requisitos funcionais

3.3 ESPECIFICAÇÃO

Conforme Bezerra (2002, p. 14) a UML é uma linguagem visual que serve para modelar sistemas orientados a objetos e é independente de linguagens de programação e processos de desenvolvimento de software. Utilizando a linguagem UML e a ferramenta Enterprise Architect foram especificados os diagramas de casos de uso, atividade, classe e seqüência.

3.3.1 DIAGRAMA DE CASOS DE USO

A Figura 4 apresenta o diagrama de casos[1] de uso da ferramenta desenvolvida.

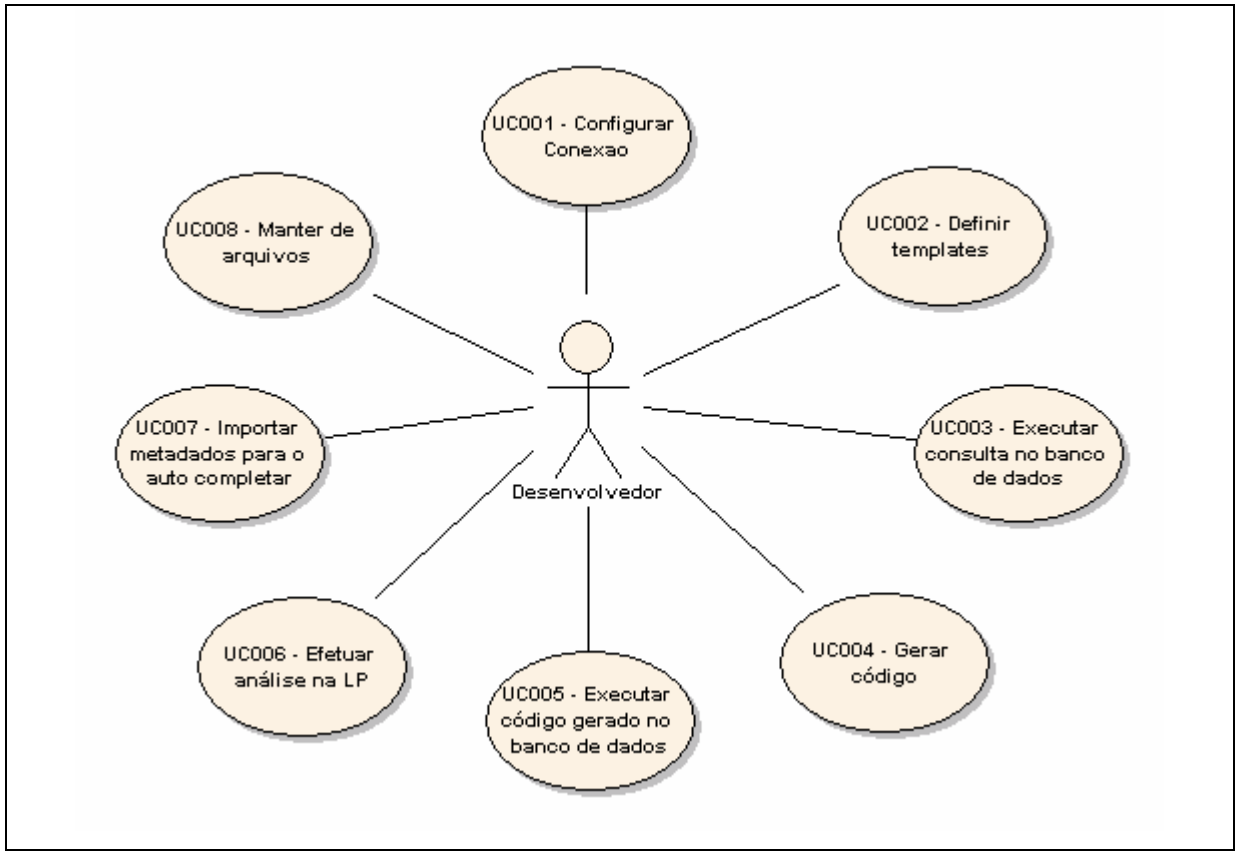


Figura 4 – Diagrama de caso de uso

A seguir são detalhados os casos de usos, apresentado a seqüência de interações entre o sistema e o usuário.

[1] O diagrama de casos de uso é um sistema composto de casos de uso, de atores e relacionamentos entre eles, o caso de uso são as interações que acontecem entre o sistema e os usuários (atores) que utilizam o sistema. O caso de uso basicamente representação narrativa do que faz e o que fazer com o sistema, desconsiderando os processos internos do sistema (BEZERRA, 2002, p. 46).

UC001 - Configurar conexão
Sumário: O desenvolvedor cria as conexões utilizando *driver* JDBC que são utilizadas para as consultas e executar *scripts* no banco de dados, as conexões são salvas em um arquivo XML.
Ator: Desenvolvedor
Fluxo principal: Criar nova conexão 1) O usuário acessa o menu configuração e sub-menu configuração conexão. 2) É apresentada a tela de configuração das conexões. 3) O usuário informa os campos do formulário. 4) O usuário clica no botão “Salvar”. 5) O sistema verifica se os campos obrigatórios foram respondidos. 6) O sistema adiciona a conexão na lista de conexões e efetua a serialização da lista em formato XML, em um arquivo do sistema. 7) O sistema disponibiliza a conexão na barra de ferramentas. 8) O sistema exibe uma mensagem informando que a conexão foi salva.
Fluxo alternativo: Abrir e testar conexão No passo 3, caso o usuário opte por abrir uma conexão e efetuar um teste de conexão 3.1.1) O usuário seleciona uma conexão na barra de ferramentas. 3.1.2) O sistema carrega as informações da conexão e preenche os campos. 3.1.3) O usuário clica no botão “Testar”. 3.1.4) O sistema testa a conexão e exibe uma mensagem informando o resultado do teste.
Fluxo alternativo: Excluir conexão No passo 3, caso o usuário opte por excluir uma conexão. 3.2.1) O usuário seleciona uma conexão na barra de ferramentas. 3.2.2) O sistema carrega as informações da conexão e preenche os campos. 3.2.3) O usuário clica no botão “Apagar” 3.2.4) O sistema verifica se tem uma conexão selecionada. 3.2.5) O sistema exibe uma mensagem de confirmação. 3.2.6) O usuário clica em “SIM” 3.2.7) O sistema retira a conexão da lista de conexões e efetua a serialização da lista.
Fluxo de exceção: Não informou campos obrigatórios No passo 5, caso o usuário não informou todos os campos obrigatórios. 5.1) O sistema exibe uma mensagem ao usuário, informando que o campo não foi informado e posiciona o cursor sobre o campo. 5.2) retorna ao passo 3.
Fluxo de exceção: Não Selecionou a conexão No passo 3.1.3 e 3.2.3, caso o usuário pule os passos e não selecione a conexão. 3.2.3.1) O sistema exibe uma mensagem informando que é obrigatório selecionar uma conexão. 3.2.3.2) retorna ao passo 3.2.1.
Pós condição: As listas de conexões foram alteradas e serializadas em um arquivo no formato XML.

Quadro 4 – Caso de uso `UC001 – Configurar conexão`

UC002 - Definir *templates*
Sumário: O desenvolvedor cria *templates* para uma determinada LP, que são utilizados na conversão das functions, também permite importar e exportar os *templates*, ao final este é serializado em formato XML e armazenados um arquivo de *templates*.
Ator: Desenvolvedor
Fluxo principal: Criar novo template 1) O usuário acessa o menu configuração e sub-menu *templates*. 2) O sistema exibe ao usuário a tela de manutenção de *templates*. 3) O usuário clica sobre o botão “Novo”. 4) O usuário informa os campos do formulário. 5) O usuário clica no botão “Salvar”. 6) O sistema verifica se os campos obrigatórios foram informados. 7) O sistema incluí a conexão na lista de *templates* e serializa a lista em formato XML em um arquivo do sistema. 8) O sistema atualiza a lista de *templates* com o novo *template*. 9) O sistema exibe uma mensagem informando que a conexão foi salva com sucesso.
Fluxo alternativo: Abrir e clonar um *template* No passo 3, caso o usuário opte por abrir um *template* e efetuar alterações. 3.1.1) O usuário seleciona um *template* na barra de ferramentas. 3.1.2) O sistema carrega as informações do *template* e preenche os campos. 3.1.3) O usuário clica no botão “Clonar”. 3.1.4) O sistema verifica se existe um template selecionado, caso existir cria um *template* idêntico ao selecionado. 3.1.4) O sistema altera o nome do *template* concatenando a palavra “(Clone)” no final.
Fluxo alternativo: Excluir *template* No passo 3, caso o usuário opte por excluir um *template*. 3.2.1) O usuário seleciona um *template* na barra de ferramentas. 3.2.2) O sistema carrega as informações do *template* e preenche os campos. 3.2.3) O usuário clica no botão “Apagar”. 3.2.4) O sistema verifica se tem um *template* selecionado. 3.2.5) O sistema exibe uma mensagem solicitando a confirmação da exclusão. 3.2.6) O usuário confirma a exclusão. 3.2.7) O sistema retira o *template* da lista de *templates* e serializa a lista de *templates* e atualiza a combo de seleção de conexões.
Fluxo alternativo: Importar *template* No passo 3, caso o usuário opte por importar um *template* já definido. 3.3.1) O usuário clica sobre o botão “importar”. 3.3.2) O sistema abre uma janela para que o usuário informe o arquivo em formato XML que conter o *template*. 3.3.3) O usuário informa qual é o arquivo que deseja importar e clica em abrir. 3.3.4) O sistema verificar se é um arquivo de *template* válido. 3.3.5) O sistema importa o *template*, atualiza a lista de *templates* e serializa a lista. 3.3.6) O sistema exibe uma mensagem que o *template* foi importado com sucesso.
Fluxo alternativo: Exportar *template* No passo 3, caso o usuário opte por exportar um *template* para um arquivo. 3.4.1) O usuário clica sobre o botão “exportar”. 3.4.2) O sistema abre uma janela para que o usuário informe a localização que será exportado o *template*.

3.4.3) O usuário informa qual local para exportar o arquivo. 3.4.4) O sistema cria o arquivo no local informado, serializa o *template* em XML e grava no arquivo.
Fluxo exceção: Não informou campos obrigatórios No passo 6, caso o usuário não informou os campos obrigatórios 6.1) O sistema exibe uma mensagem que os campos obrigatórios não foram informados. 6.2) Retorna ao passo 4.
Fluxo exceção: Não selecionou template Nos passos 3.2.4, 3.4.2, caso o usuário não tenha selecionado um *template*. 6.1) O sistema exibe uma mensagem informando que é obrigatório a seleção de um *template*. 6.2) Retorna ao passo anterior.
Fluxo exceção: *Template* inválido Nos passos 3.3.4, caso o usuário informe um *template* inválido. 6.1) O sistema exibe uma mensagem que o *template* informado é inválido. 6.2) Retorna ao passo 3.
Pós condição: As listas de *template* foi alterada. *Templates* foram personalizados. *Templates* foram importados ou exportados. *Templates* foram serializados em arquivo XML.

Quadro 5 – Caso de uso UC002 - Definir *templates*

UC003 - Executar consulta no banco de dados
Sumário: O desenvolvedor pode executar comandos `SELECT` no banco de dados e visualizar seu resultado. O *script* é executado utilizando a conexão selecionada na barra de ferramentas.
Ator: Desenvolvedor
Pré-condição: Uma conexão deve ter sido cadastrada no sistema e estar funcionando corretamente.
Fluxo principal: Executar consulta no banco de dados 1) O usuário seleciona nos sistema a aba de SQL. 2) O sistema exibe a tela, com um campo que permite desenvolver comandos `SELECT`. 3) O usuário cria uma consulta em formato SQL. 4) O usuário seleciona a conexão que será executada a consulta e clica em executar. 5) O sistema efetua a conexão com o banco de dados. 6) O sistema executa a consulta no banco de dados e monta o resultado em uma tabela na tela.
Fluxo de exceção: Não selecionou a conexão No passo 4, caso o usuário não tenha selecionado uma conexão ou não tenha nenhuma conexão criada no sistema. 4.1) O sistema exibe uma mensagem que não foi possível conectar ao banco de dados. 4.2) Retorna ao passo 2.
Fluxo de exceção: Não informou comando de consulta No passo 4, caso o usuário não tenha informado o comando de consulta 4.1) O sistema exibe uma mensagem de erro durante a execução. 4.2) Retorna ao passo 2.
Fluxo de exceção: Consulta informa contem erros No passo 6, caso o usuário tenha criado um comando de consulta que contenha erros 4.1) O sistema exibe uma mensagem que ocorreu um erro durante a execução e exibe as informações retornadas pelo banco. 4.2) Retorna ao passo 2.
Pós condição: As consultadas foram executas no banco de dados e os resultados exibidos ao usuário.

Quadro 6 – Caso de uso `UC003 - Executar consulta no banco de dados`

UC004 - Gerar código
Sumário: O desenvolvedor pode efetuar a geração da LP implementada no editor, com os *templates* personalizados.
Ator: Desenvolvedor
Pré-condição: Possuir *templates* criados. Possuir LP desenvolvida no editor.
Fluxo principal: Geração de código 1) O usuário seleciona o menu conversão e o sub-menu gerar. 2) O sistema efetua a verificação de erro no *script* da LP. 3) O sistema seleciona um *template* e efetua a geração utilizando ações semânticas e a API *Velocity*. 4) O sistema após gerar um *template* adiciona o conteúdo gerado, na área de texto da aba correspondente ao *template*. 5) O sistema exibe uma mensagem que os *templates* foram gerados com sucesso.
Fluxo de exceção: Encontrou erro no *script* da LP No passo 2, caso o sistema encontre erro na LP 2.1) O sistema exibe a aba do editor, mostra a área de erros com a mensagem do erro encontrado. 2.2) Retorna ao passo 1.
Pós condição: Foi gerado novos *scripts* para os *templates* personalizados.

Quadro 7 – Caso de uso `UC004 – Gerar código`

UC005 - Executar código gerado no banco de dados
Sumário: O desenvolvedor pode executar as LPs geradas a partir dos *templates* diretamente nos bancos de dados.
Ator: Desenvolvedor
Pré-condição: Ter uma conexão cadastrada no sistema e estar funcionado. Ter um *template* cadastrado no sistema. Ter uma LP desenvolvida e gerada utilizando um *template*.
Fluxo principal: Executar LP 1) O usuário clica na aba de um *template* que tenha sido efetuada a conversão. 2) O sistema exibe o *script* do *template*. 3) O usuário seleciona uma conexão na barra de ferramentas que corresponda ao banco que deseja inserir a LP. 4) O usuário clica no botão “executar function“. 5) O sistema verifica se tem uma conexão selecionada. 6) O sistema efetua a conexão e executa o *script* da LP. 7) O sistema exibe uma mensagem que a LP foi executada com sucesso.
Fluxo de exceção: Não selecionou a conexão No passo 5, caso o usuário não tenha selecionado uma conexão ou não tenha nenhuma conexão criada no sistema. 5.1) O sistema exibe uma mensagem informando que é obrigatório selecionar uma conexão. 5.2) Retorna ao passo 1.
Fluxo de exceção: Problema ao executar LP No passo 6, caso ocorrer problema ao executar uma LP 6.1) O sistema exibe uma mensagem como o erro que ocorreu durante a execução. 6.2) retorna ao passo 1.
Pós condição: A LP selecionada é inserida ou compilada no banco de dados correspondente a conexão selecionada.

Quadro 8 – Caso de uso `UC005 – Inserir LP no banco de dados`

UC006 - Efetuar análise da LP
Sumário: O desenvolvedor pode verificar os possíveis erros ocorridos durante a criação da LP e o sistema efetua análise léxica e sintática da LP desenvolvida.
Ator: Desenvolvedor
Fluxo principal: Efetuar análise da LP 1) O usuário cria uma *function* utilizando o editor e suas funcionalidades. 2) O usuário clica no menu conversão, sub-menu validar. 3) O sistema verifica se o código é valido executando a análise léxica e sintática. 4) O sistema exibe uma mensagem informando que a LP está correta e pronta para ser gerada.
Fluxo de exceção: Encontrado erro durando a análise No passo 3, caso o usuário tenha cometido algum erro léxico ou sintático. 3.2.1) O sistema encontra o erro na LP e gera uma exceção. 3.2.2) O sistema exibe a área de mensagem de erro, inclui a mensagem e configura para quando o usuário clicar duas vezes sobre o erro posicionar o cursor onde foi encontrado o erro.
Pós condição: Efetuado análise e informado ao usuário se a LP esta escrita de forma correta, caso encontrar erros exibe mensagem de erro e auxilia o usuário a encontrar o erro.

Quadro 9 – Caso de uso `UC007 - Efetuar análise da LP`

UC007 - Importar metadados para auto complemento
Sumário: O desenvolvedor pode incluir na funcionalidade de auto completar, o nome de todas as tabelas e colunas das tabelas do banco de dados para auxiliá-lo e agilizar o processo de desenvolvimento da LP.
Ator: Desenvolvedor
Pré-condição: Ter uma conexão cadastrada no sistema e estar funcionando corretamente. Uma conexão deve estar selecionada e este banco possuir tabelas.
Fluxo principal: Importar metadados do banco de dados. 1) O usuário seleciona uma conexão. 2) O usuário clica no botão para atualizar auto completar. 3) O sistema verifica se tem uma conexão selecionada. 4) O sistema conecta ao banco de dados, lê o metadados do banco utilizando API JDBC. 5) O sistema inclui na lista de auto completar das tabelas e colunas encontradas no metadados. 6) O sistema armazena em uma lista na memória os itens incluídos na lista de auto completar.
Fluxo alternativo: Executou o auto completar em outro banco No passo 5, caso o usuário já tenha importado o metadados de outro banco de dados. 5.1) O sistema verifica a lista de itens armazenada na memória, se a mesma contem itens, caso conter remove-os da lista de auto completar. 5.2) Retorna ao passo 4.
Fluxo de exceção: Não selecionou a conexão No passo 3, caso o usuário não tenha selecionado a conexão ou não tenha nenhuma conexão no criada no sistema 3.1) O sistema exibe uma mensagem que é obrigatório selecionar uma conexão. 3.2) Retorna ao passo 1.
Fluxo de exceção: Erro durando a obtenção dos dados. No passo 4, caso o sistema encontre algum erro durante a conexão ou a obtenção do metadados. 4.1) O sistema exibe uma mensagem de erro durante a execução e juntamente com o erro retornado do banco de dados ou da aplicação. 4.2) Retorna ao passo 1.
Pós condição: Novos itens foram incluídos na lista de auto completar.

Quadro 10 – Caso de uso `UC007 - Importar metadados para auto completar`

UC008 - Manter de arquivos
Sumário: O desenvolvedor tem a possibilidade de salvar, salvar como e abrir arquivo com *scripts* das LPs.
Ator: Desenvolvedor
Pré-condição: Possuir arquivo salvos ou LPs escritas.
Fluxo principal: Abrir LP desenvolvida. 1) O usuário seleciona o menu arquivo. 2) O usuário clica sobre o sub-menu “abrir”. 3) O sistema exibe uma mensagem pergunta se o usuário deseja salvar *script* da LP. 4) O usuário informa faz sua escolha, caso escolher sim vai para 2.1.1. 5) O sistema exibe uma janela para que seja selecionado o arquivo com o *script* da LP. 6) O usuário seleciona o arquivo com o *script* LP. 7) O sistema efetua a leitura do arquivo e exibe seu conteúdo no editor. 8) O editor aplica a funcionalidade de sintaxe *highlight*.
Fluxo alternativo: Salvar o arquivo No passo 2, caso o usuário opte por salvar o arquivo. 2.1.1) O sistema verifica se o arquivo já está salvo, caso não esteja exibe uma janela para a que seja informado o local de destino do *script* da LP. 2.1.2) O usuário seleciona o local que deseja salvar o *script* e clica no botão “salvar”. 2.1.3) O sistema verifica se o arquivo já existe, caso não existir cria e inclui o conteúdo no arquivo. 2.1.4) Retorna ao passo 1.
Fluxo alternativo: Salvar como o arquivo No passo 2, caso o usuário opte por salvar o arquivo em outro local. 2.2.1) O sistema exibe uma janela para a que seja informado o local de destino do *script* da LP. 2.2.2) O usuário seleciona o local que deseja salvar o *script* e clica no botão “salvar”. 2.2.3) O sistema verifica se existe o arquivo, caso não existir cria e inclui o conteúdo. 2.2.4) Retorna ao passo 1.
Pós condição: Arquivos foram exibidos, alterados ou salvos utilizando o sistema.

Quadro 11 – Caso de uso `UC008 - Manutenção de arquivos`

3.3.2 DIAGRAMA DE ATIVIDADES

A Figura 5 representa o diagrama de atividade do sistema[2], que descreve os processos de criação, importação e exportação de *templates*.

[2] O diagrama de atividade é um diagrama orientado a fluxo de controle parecido com um fluxograma, sendo um tipo especial do diagrama de estados, em que são representados os estados de uma atividade (BEZERRA, 2002, p. 228).

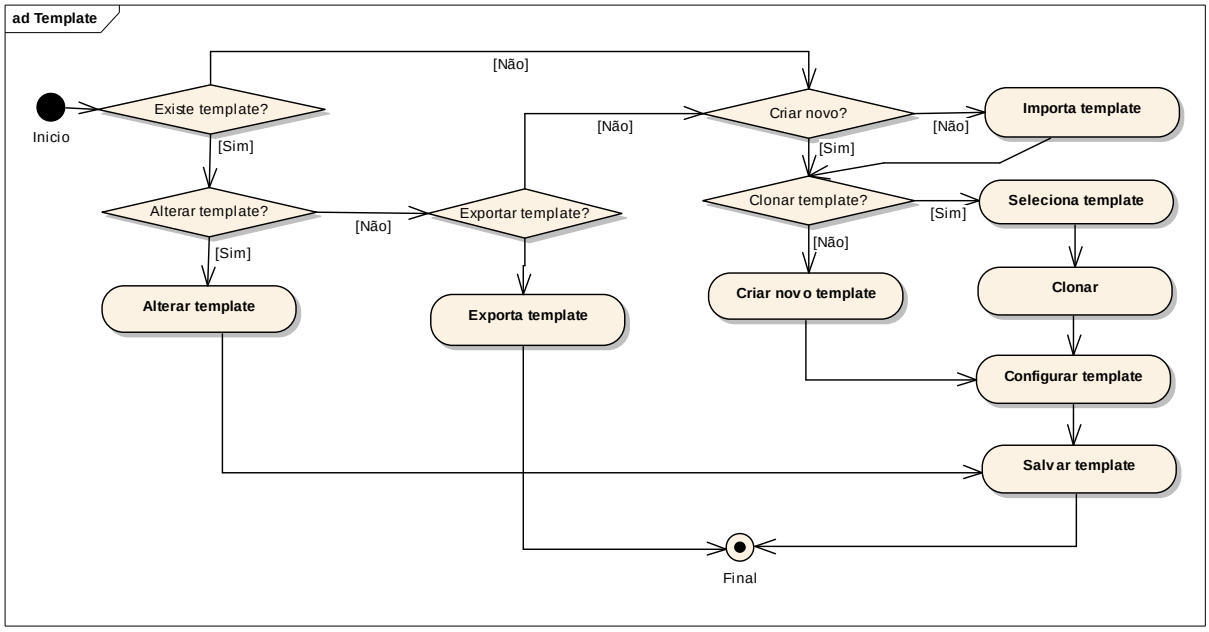


Figura 5 – Diagrama de atividades

Para utilizar o ambiente inicialmente é necessário possuir *templates* customizados para as LPs.Adicionar *templates* ao ambiente existem duas formas: criar novos *templates* ou importar a partir de um *template* previamente criado e exportado.

Para criação de novos *templates* é necessário preencher todas as configurações solicitadas na tela. Todos os campos possuem um botão de ajuda ao lado direito para auxiliar o preenchimento do campo. As LPs que não possuírem funcionalidade correspondentes às dos *templates* devem ser deixado sem preenchimento, como por exemplo a opção “DBO.” que é somente utilizada no SQL Server. Para preencher os campos existem duas formas: substituição e *templates* Velocity.

A interpretação dos comandos da linguagem padrão pode ser feita a partir uma substituição do comando, pelo código contido no *template*, onde o campo simplesmente é substituído pelo seu correspondente, exemplo o campo “varchar” ser substituído por “varchar”. A na utilização de *templates* Velocity é semelhante a anterior, mas com funcionalidades de campos coringas. Campos coringas são utilizados em expressão que contém algo em comum, como por exemplo a instrução de condição *if*, que customizado como *template*: “if ($CONDICAO)”, pode ser declarado de várias formas mas todas têm em comum a condição de teste.

A Figura 6 apresenta o digrama de atividade com os processos de validação e geração de código da LP.

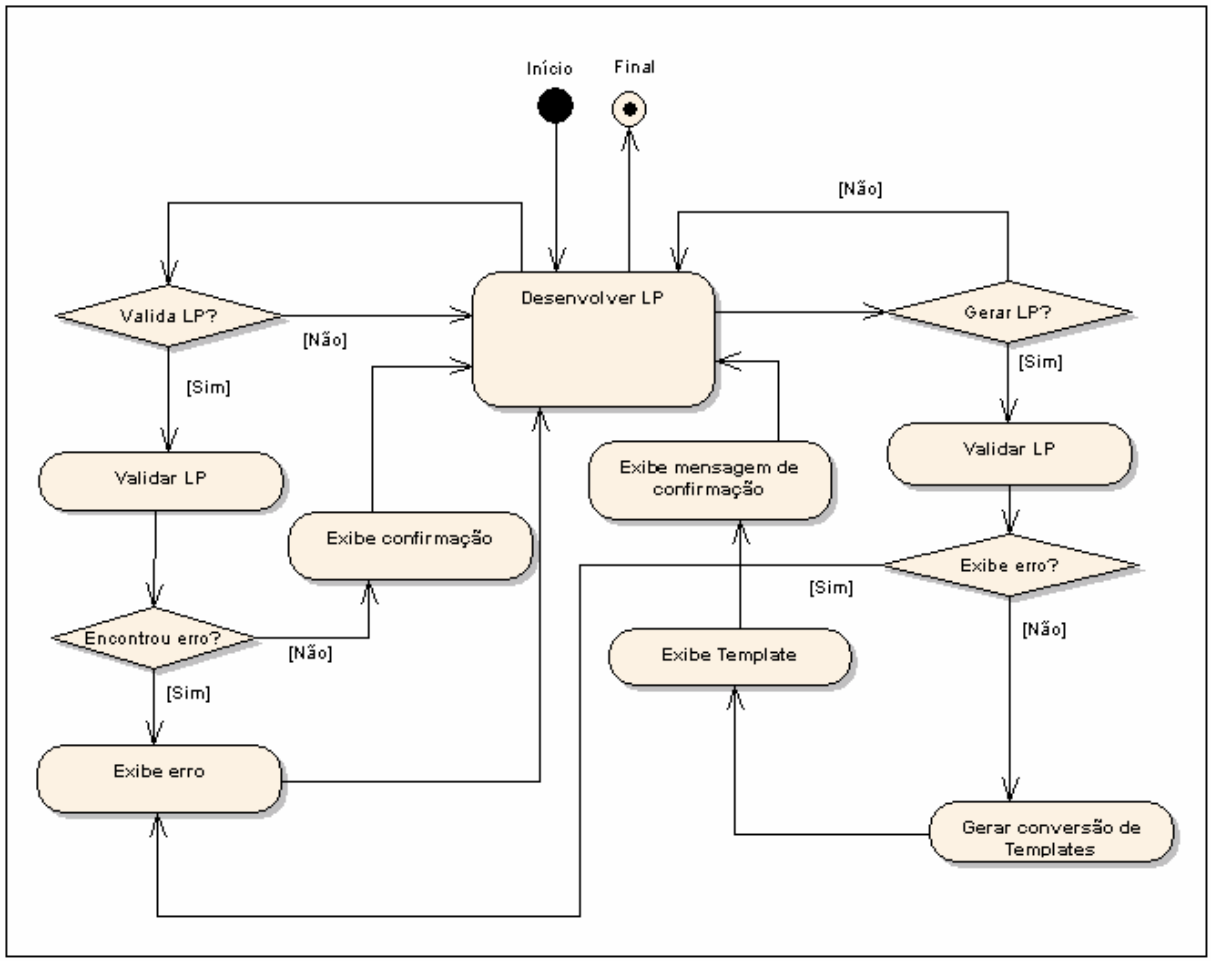


Figura 6 - Validação e geração de código

A validação da linguagem efetua as análises léxicas, sintéticas e semânticas, avaliando a linguagem em busca de possíveis erros que podem ter acontecido no desenvolvimento. Durante a validação caso for encontrado alguma insistência é disparado um aviso com informações sobre o erro.

O processo de geração de código efetua todo o processo de validação e após converte a linguagem a partir de *templates*. Ao termino do processo exibe os resultados da geração e uma mensagem de confirmação.

3.3.3 DIAGRAMA DE CLASSES

A Figura 7 mostra o diagrama de classes[3] do pacote `modelo.template`, que é responsável por possuir a implementação dos *templates*. Todas as classes deste pacote

[3] O diagrama de classes é utilizado para descrever os aspectos estruturais estáticos de um sistema orientado a objetos, onde descreve as classes com seus métodos, atributos e os relacionamentos entre as classes, o digrama de classe evolui com o desenvolvimento do sistema.

implementam a interface `java.io.Serializable`, possibilitando assim que os objetos sejam persistidos em disco. A classe `BancoDados` implementa a inteface `java.lang.Clonable`, que permite que um objeto seja clonado, ou seja, é possível criar uma cópia fiel do objeto.

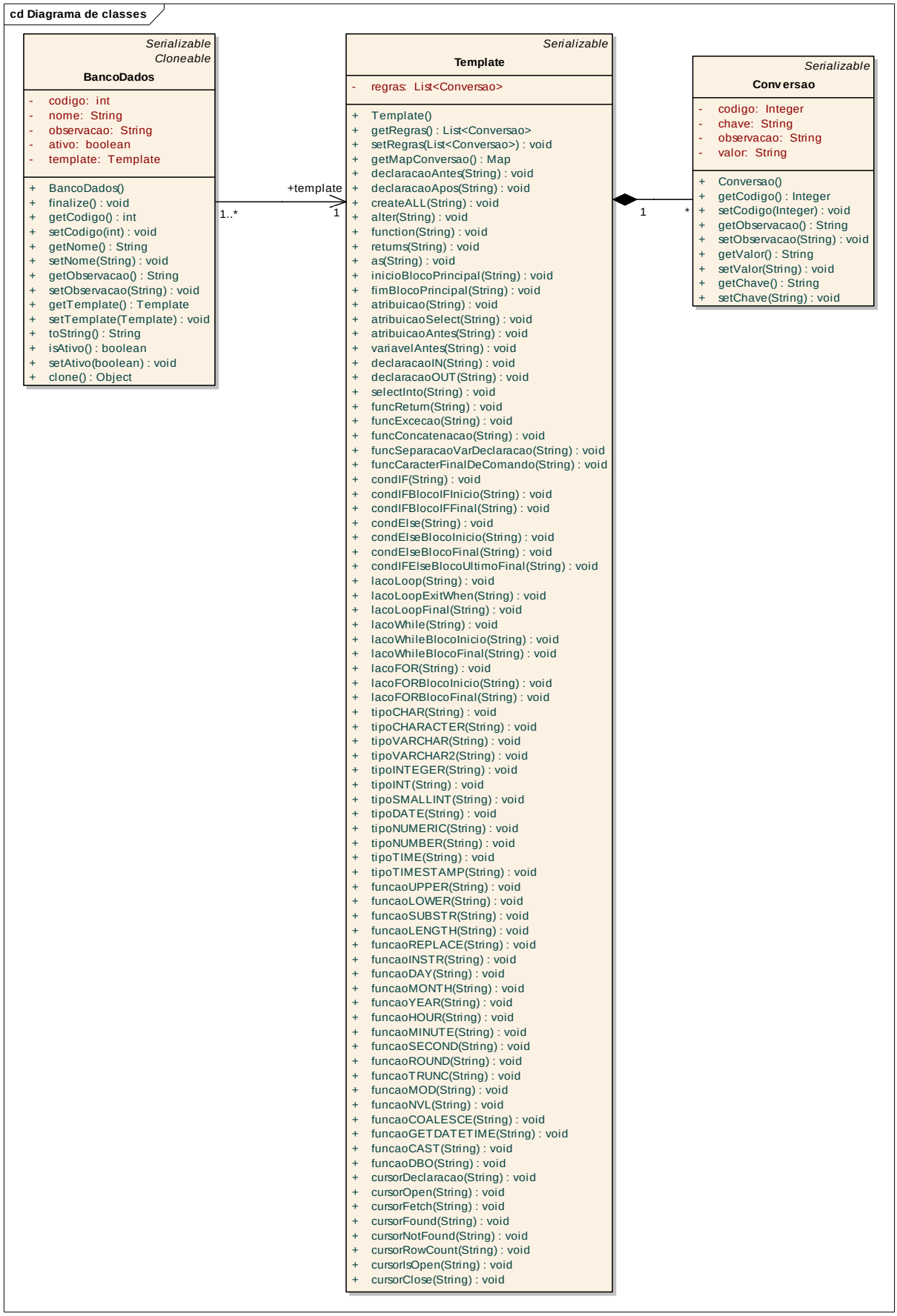


Figura 7 - Diagrama de classes dos *templates*

As classes apresentadas na Figura 7 são:

a) `BancoDados`: classe responsável por armazenar as informações do *template* como o nome, se o status está ativo, e um *template*;

b) `Template`: classe responsável por armazenar a lista de propriedades de cada *template*;

c) `Conversao`: classe responsável por definir quais as propriedades que cada ação dos *templates* possui, que são chave para identificar a ação, uma observação que é a ajuda e um valor que é onde ficam armazenadas as informações que o usuário informou.

A Figura 8 representa o diagrama de classe da funcionalidade de auto complemento de palavras no editor. As classes `WordMenuWindows` e `Word` são classes internas a classe `PowerEditor` e as classes `WordMenuLeyListener` e `WordMenuMouseListner` são classes internas das classe `WordMenuWindows`.

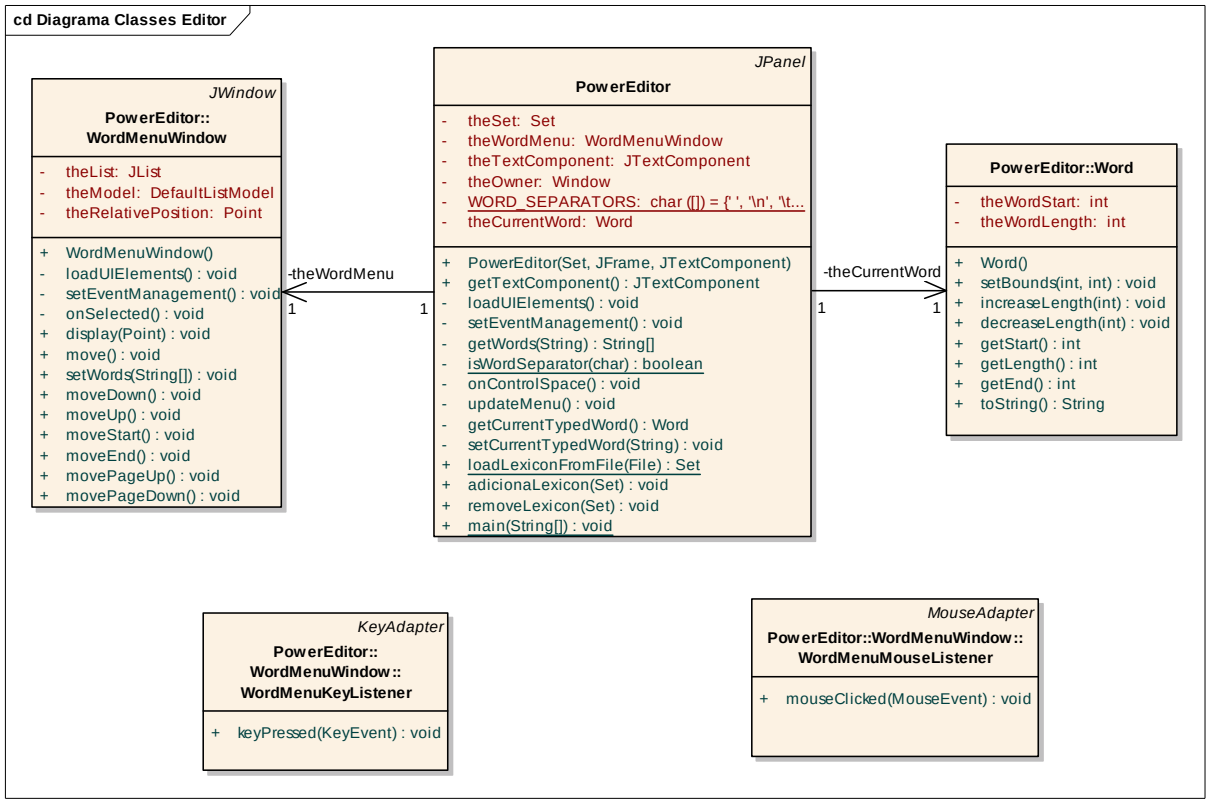


Figura 8 - Diagrama de classes do editor

As classes apresentadas na Figura 8 são:

a) `PowerEditor`: classe responsável por gerenciar e selecionar a lista de opções que são exibidas e adicionar os itens selecionado no editor;

b) `WordMenuWindow`: classe responsável por manter o menu com as opções de complemento e mover a seleção de itens;

c) `Word`: responsável por gerenciar a palavra que está sendo complementada e manter as informações sobre a localização;

d) `WordMenuMouseListener`: responsável por atender os eventos no mouse no menu de itens;

e) `WordMenuKeyListener`: responsável pela ações vindas no teclado sobre o menu de itens.

A Figura 9 representa o diagrama de classes da funcionalidade de sintaxe *highlight* das palavras no editor. As classes `HighlightDocument` e `NoWrappingTextPane` são classes internas a classe `Highligter`. A classe `MeuPlToken` implementa a classe `IsyntaxColor`.

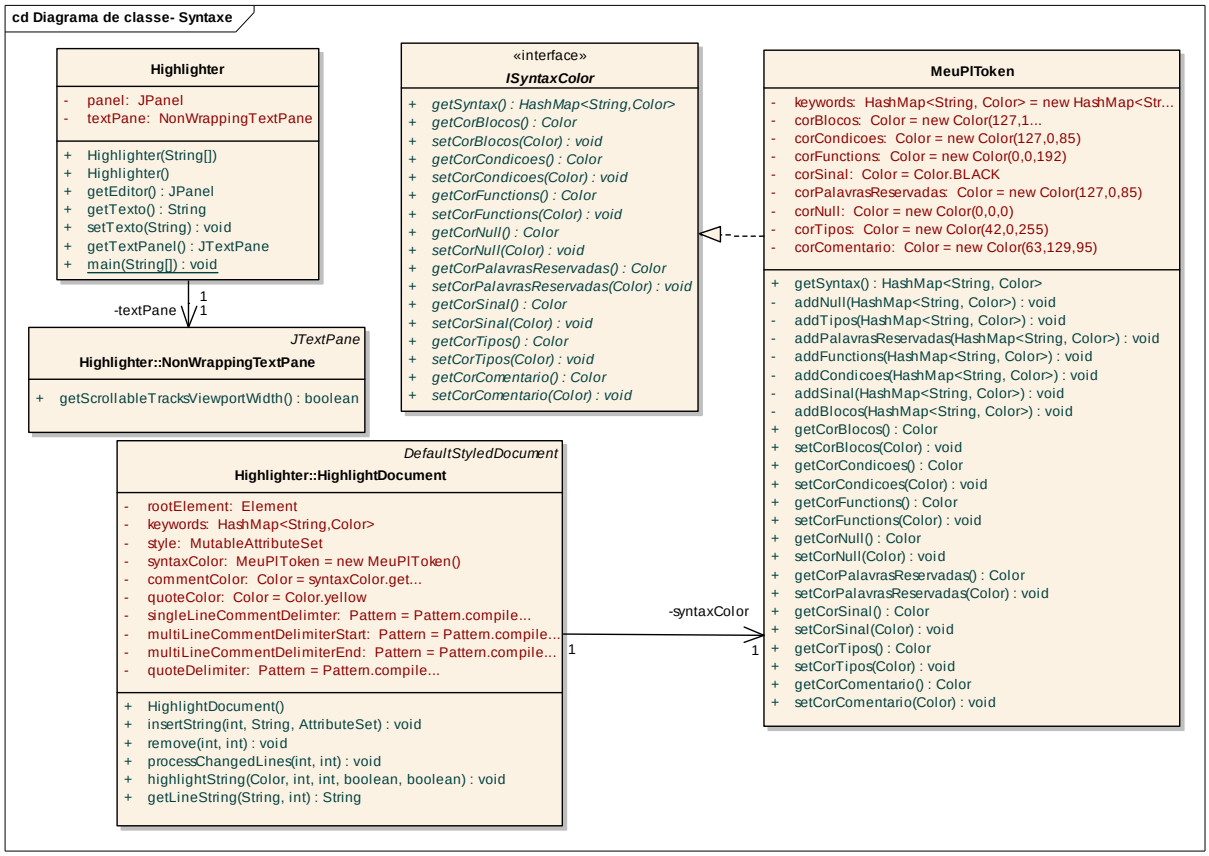


Figura 9 - Diagrama de classes da sintaxe

As classes apresentadas na Figura 9 são:

a) `Highlighter`: responsável por gerenciar o editor;

b) `NonWapppingTextPan`: responsável pela área de texto do editor;

c) `HightlightDocument`: classe responsável por alterar as cores das palavras conforme uma lista de palavras e cores já pré-definidas;

d) `ISyntaxColor`: a interface é utilizada para padronizar as sintaxes e facilitar a criação de novas sintaxes, nela são definidos os métodos que são necessários

implementar;

e) `MeuPlToken`: classe é responsável por manter a lista de palavras que recebem cores diferentes e sua cores.

A Figura 10 representa o diagrama de classes da funcionalidade de geração de código. Neste diagrama as classes que tratam exceções e as de constantes não foram incluídas para uma melhor visualização.

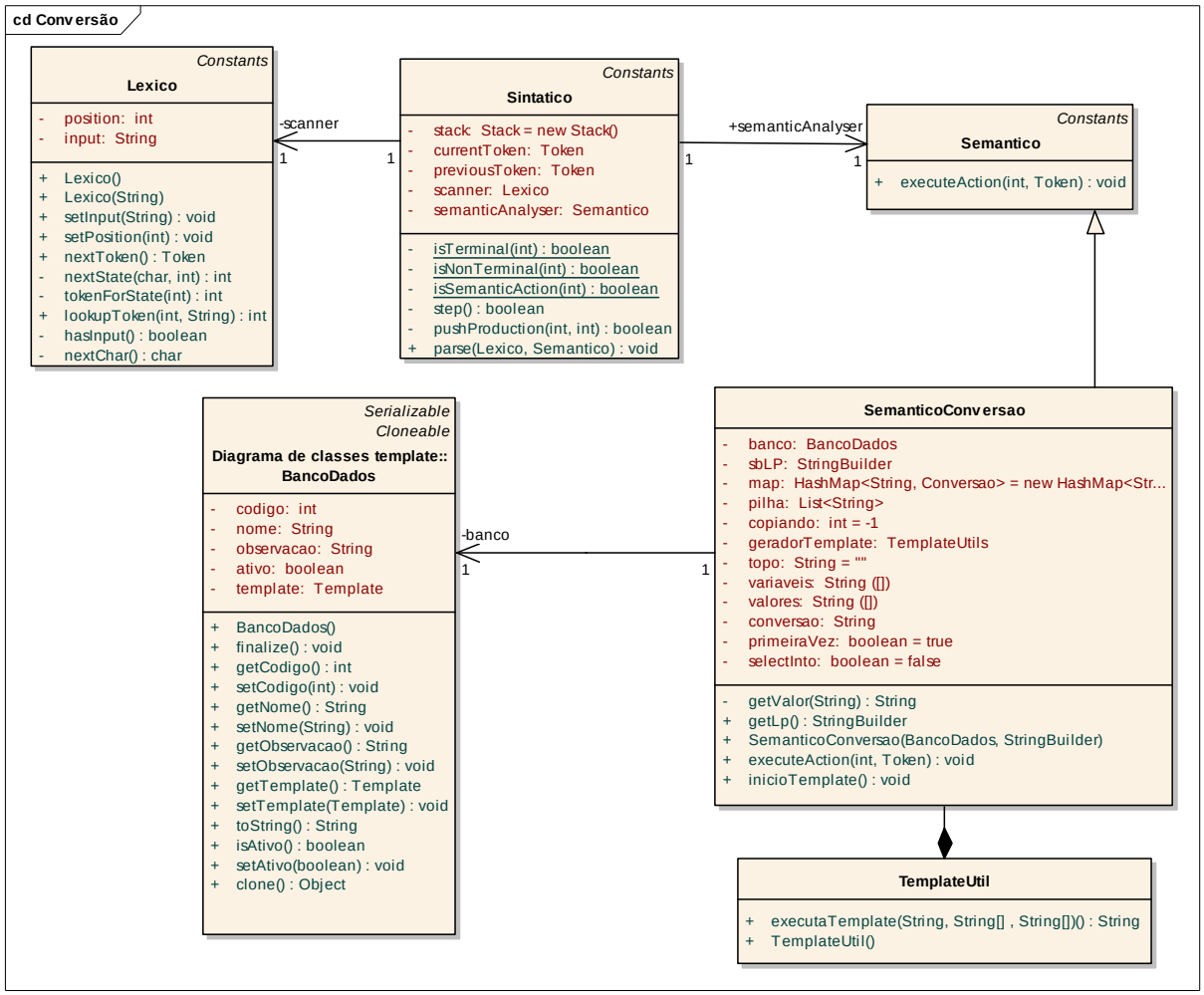


Figura 10 - Diagrama de classes da geração de código

As classes apresentadas na Figura 10 são:

f) `Lexico`: responsável pela análise léxica da LP;

g) `Sematico`: responsável pela análise semântica da LP;

h) `Semantico`: contém os métodos que são chamados para resolver as ações semânticas;

i) `SemanticoConversao`: responsável pela conversão das ações semânticas conforme os *templates*;

j) `BancoDados`: contém o *template* utilizado na geração de código da LP;

k) `TemplateUtis`: classe utilitária utilizada para executar as ações semântica que utilizam o Velocity.

3.3.4 DIAGRAMA DE SEQÜÊNCIA

A Figura 11 apresenta o digrama de seqüência do processo de conversão da LP implementada pelo desenvolvedor e convertida utilizando *templates* previamente customizado.

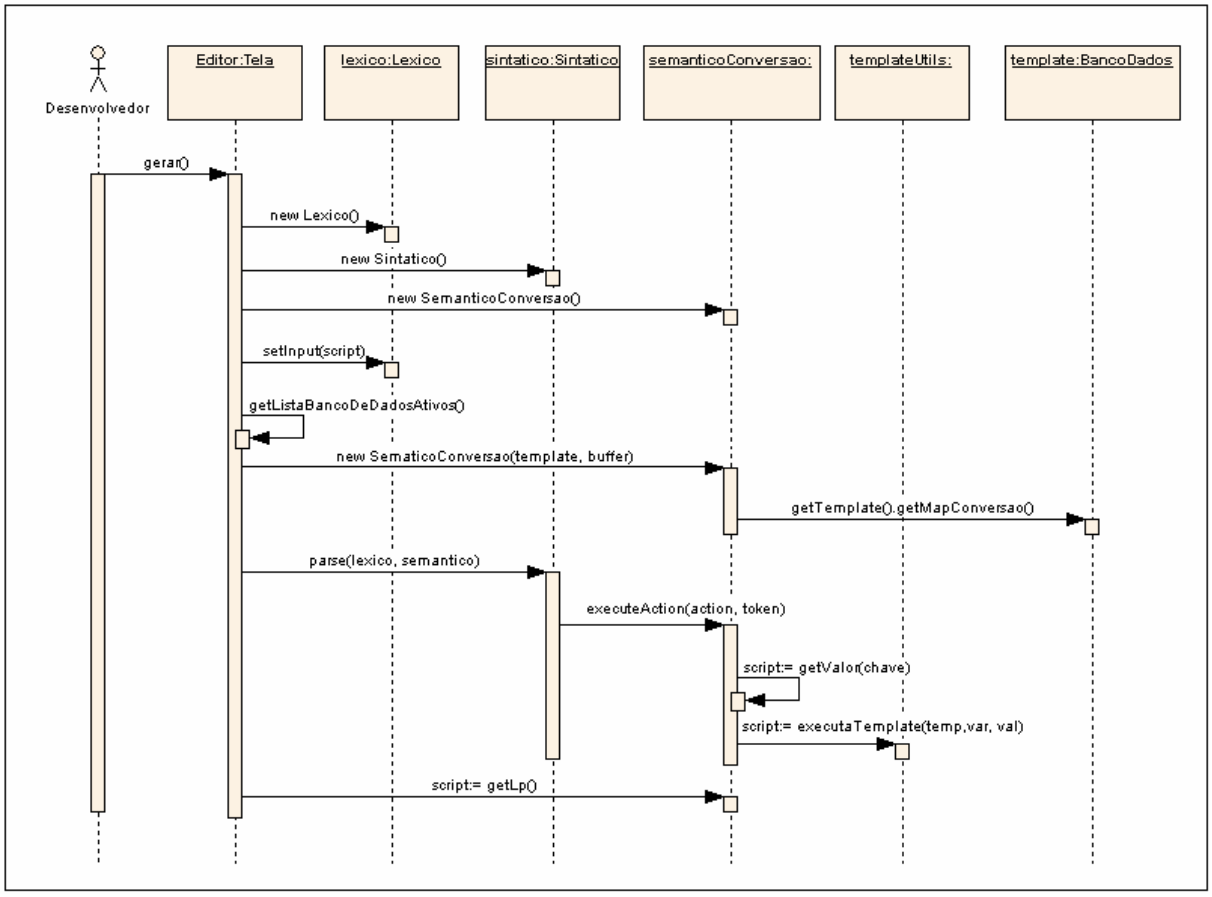


Figura 11 - Diagrama de seqüência

O diagrama de seqüência apresentado na Figura 11 apresenta a seqüência de execução da geração de fontes, nele são instanciados os analisadores léxico, sintático e semântico. O analisador semântico é responsável pela iteração com o *template* selecionado e a execução das chamadas do Velocity, que são executados na classe estática `TemplateUtils`.

3.4 IMPLEMENTAÇÃO

Nesta sessão são mostradas as técnicas utilizadas para a implementação, ferramentas que auxiliaram o desenvolvimento e o funcionamento do ambiente implementado.

3.4.1 TÉCNICAS E FERRAMENTAS UTILIZADAS

O ambiente apresentado foi desenvolvido em linguagem Java, utilizando o ambiente de desenvolvimento NetBeans 6.0, pois apresenta facilidades para trabalhar com as bibliotecas gráficas que foram utilizadas, são elas o SWING, AWT e `swinglabs-0.8.0`, JDK versão 1.6.0.

Para geração dos analisadores léxico e sintático foi utilizada a ferramenta GALS versão 2003.10.03. A implementação do analisador semântico, não é totalmente contemplada pela ferramenta, mas são disponibilizadas ações semânticas, que são executadas pelo analisador sintático, permitindo a implementação do analisador para mais informações sobre a BNF veja o Apêndice A . Nas ações foi utilizado o motor de *templates* Velocity, através da biblioteca `velocity-dep-1.5.jar`. O Quadro 12 mostra a classe `TemplateUtils` para exemplificar como é utilizada a conversão de *templates*.

```java
public class TemplateUtils {

    /**
     * Execução de templates
     * @param template String com o template
     * @param variaveis variáveis contidas no template
     * @param valores valores que deve ser atribuidos as variáveis
     * @return texto convertido
     */
    public String executaTemplate(String template, String[] variaveis, String[] valores){
        try {
            //inicializa o Velocity
            Velocity.init();
            //cria o Contexto
            VelocityContext context = new VelocityContext();
            //atribui os valores as veriáveis
            for (int i = 0; i < variaveis.length; i++) {
                context.put(variaveis[i], valores[i]);
            }

            StringWriter w = new StringWriter();
            //executa o template
            Velocity.evaluate( context, w, "./executa.vm", template );
            return w.toString();
        } catch (Exception ex) {
            System.out.println("Erro durante a execução do template:"+ ex.getMessage());
        }
        //caso gerar algum erro retorna um string vazia.
        return "";
    }
}
```

Quadro 12 - Classe `TemplateUtils`

Na serialização dos dados em formato XML foram utilizadas as bibliotecas `xstream-1.0.1.jar` e `xpp3-1.1.3.3_min.jar`, por facilitarem na persistência das informações e serializar em formato XML, permitindo assim a possibilidade de *templates* serem implementados fora do ambiente se necessário. O Quadro 13 e o Quadro 14 exibe a utilização da biblioteca para exportação de *templates* e importação respectivamente.

```
public static BancoDados importaTemplate(String file){
    BancoDados banco = null;
    try{
        //efetua abertura do arquivo
        FileReader filer = new FileReader(file);
        // cria um buffer de leitura do arquivo
        BufferedReader bufRed = new BufferedReader(filer);
        //armazenar as informações do arquivo
        StringBuilder sb = new StringBuilder();
        String s;
        // recupera as informações do arquivo linha por linha
        do{
            s = bufRed.readLine();
            if(s != null){
                sb.append(s);
            }
        }while (s != null);

        //verifica se não é um arquivo vazio
        if(!sb.toString().equals("")){
            XStream xstream = new XStream();
            //cria os alias para as classes, para melhor visualização
            xstream.alias("banco", BancoDados.class);
            xstream.alias("template",Template.class);
            xstream.alias("conversao", Conversao.class);
            //a partir do arquivo carregado gera um objeto BancoDados
            banco = (BancoDados) xstream.fromXML(sb.toString());
        }
        //fecha o arquivo
        bufRed.close();
        filer.close();

    }catch(IOException e){
        System.out.println(e.getMessage());
        return null;
    }

    return banco;
}
```

Quadro 13 - Importação de *templates*

```
public static String exportar(Object p, String file){
    try{
        //verifica se o arquivo não existe então cria
        File arquivo = new File(file);
        if(!arquivo.exists()){
            if(!arquivo.createNewFile()){
                return "Arquivo não pode ser criado";
            }
        }
        FileWriter filew = new FileWriter(file);
        XStream xstream = new XStream();
        //cria os alias para as classes
        xstream.alias("banco", BancoDados.class);
        xstream.alias("template",Template.class);
        xstream.alias("conversao", Conversao.class);
        //converte o objeto para XML
        filew.append(xstream.toXML(p));
        //grava e fecha o arquivo.
        filew.flush();
        filew.close();

    }catch(IOException e){
        System.out.println(e.getMessage());
        return e.getMessage();
    }
    return null;
}
```

Quadro 14 - Exportar *templates*

As rotinas de obtenção dos metadados, consultas a partir de comandos `SELECT` e execução das LPs foram feitas através da API JDBC, conforme exemplo de conexão no Quadro 15. As conexões com os SGBDs SQL Server e Oracle foram estabelecidas utilizando as APIs `jtds-1.2.2.jar` e `ojdbc14.jar` respectivamente.

```
public static Set getTabelaEColunaDaConexao(Conexao con){
    //set para armazenar lista de tabelas e colunas
    Set tabelasEColunas = new TreeSet();
    try {
        //carrega driver
        java.lang.Class.forName(con.getDriver());
        //conecta ao banco de dados
        Connection conexao = null;
        if("".equals(con.getUsuario().trim())){
            conexao = DriverManager.getConnection(con.getUrl());
        }else{
            conexao = DriverManager.getConnection(con.getUrl(),
                                con.getUsuario(), con.getSenha());
        }
        //seleciona quais os tipos de objetos recuperar do banco
        String[] tipos = new String[1];
        tipos[0] = "TABLE";

        //recupera o metadados
        DatabaseMetaData metaData = conexao.getMetaData();
        //recupera as tabelas
        ResultSet resultSet = metaData.getTables(null, null, "%", tipos);
        while(resultSet.next()){
            String nomeTabela = resultSet.getString(3);
            tabelasEColunas.add(nomeColuna);
        }
        resColuna.close();
    }
    resultSet.close();
    conexao.close();
} catch (ClassNotFoundException ex) {
    JOptionPane.showMessageDialog(null, "Ocorreu erro durante a " +
    "obtenção dos dados\n"+ex.getMessage(), "Erro",
    JOptionPane.ERROR_MESSAGE);
} catch (SQLException ex) {
    JOptionPane.showMessageDialog(null, ex.getMessage(), "Erro",
            JOptionPane.ERROR_MESSAGE);
}
return tabelasEColunas;
```

Quadro 15 - Método que recupera as tabelas e colunas do SGBD

O ambiente permite conexões para outros SBGDs utilizando *drivers* JDBC. Após ser iniciado o sistema inclui os *drivers* contidos na pasta JDBC na raiz do sistema no *classpath* da aplicação.

Os testes foram feitos utilizando os SGBDs Oracle XE e SQL Server 2005 Express Edition, onde foi especificado modelagens de bancos de dados relacional para serem utilizado em testes da aplicação.

3.4.2 OPERACIONALIDADE DA IMPLEMENTAÇÃO

Ao executar a ferramenta, será apresentada ao usuário a tela principal do Ambiente, conforme Figura 12. Ao iniciar o sistema é apresentada a aba do editor (1) a partir desta tela o desenvolvedor tem acesso a várias funcionalidades do ambiente, como efetuar consultas SQL (2), visualizar os *templates* implementados (3), validação de código fonte (4), geração de códigos, selecionar as conexões configuradas (6), executar *scripts* caso estiver em uma aba de *templates* (7), importar auto completar (8) e ter acesso a botões padrão de novo, abrir e salvar arquivos, bem como os de acesso a área de transferência que são os botões de recortar, copiar e colar.

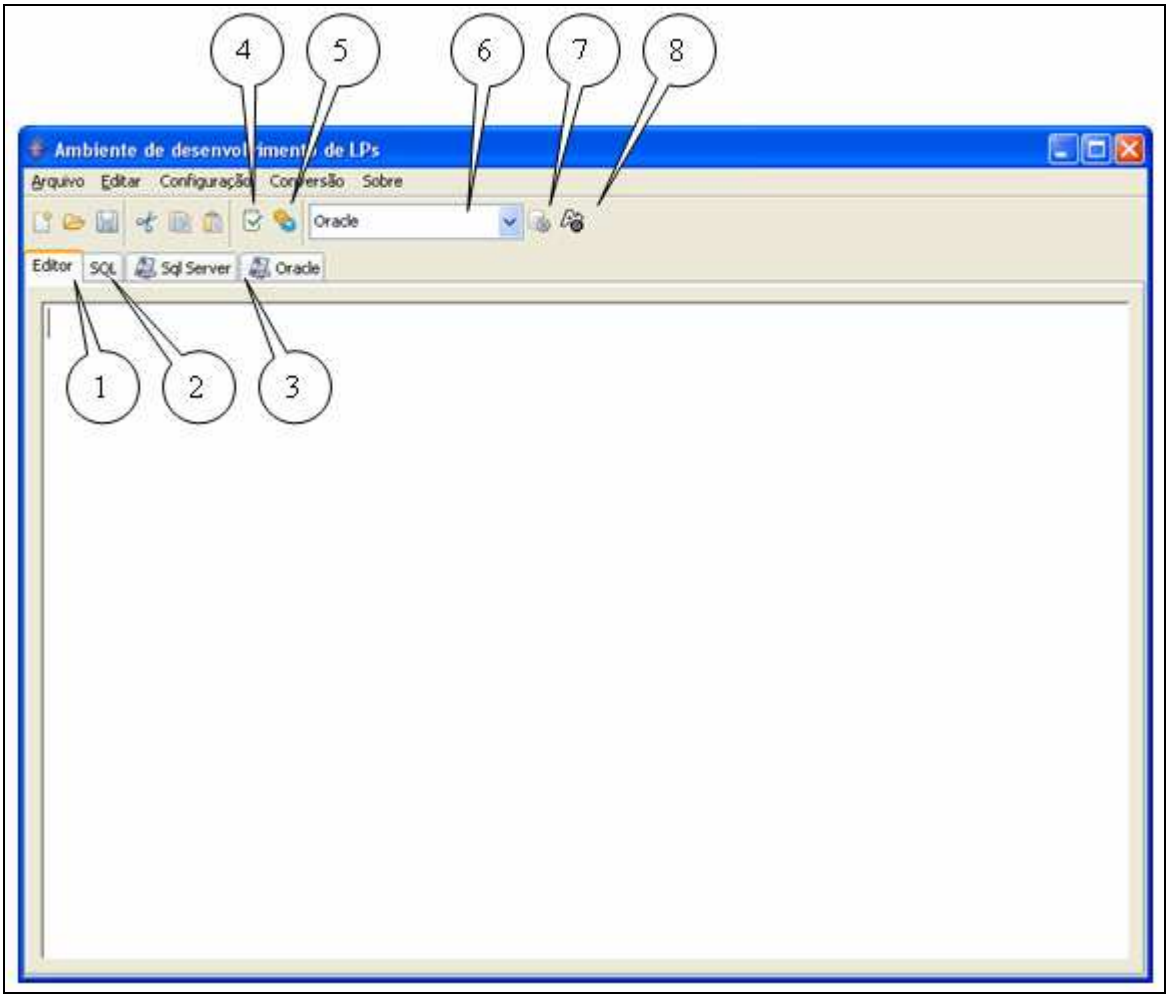


Figura 12 - Ambiente de desenvolvimento

A Figura 13 demonstra o tratamento de erros na verificação ou geração da LP. Ao encontrar um erro o ambiente exibe uma área informando o erro encontrado, para localizar o erro, basta efetuar um duplo clique sobre a mensagem ou a imagem “X” que o cursor será posicionado no local do erro.

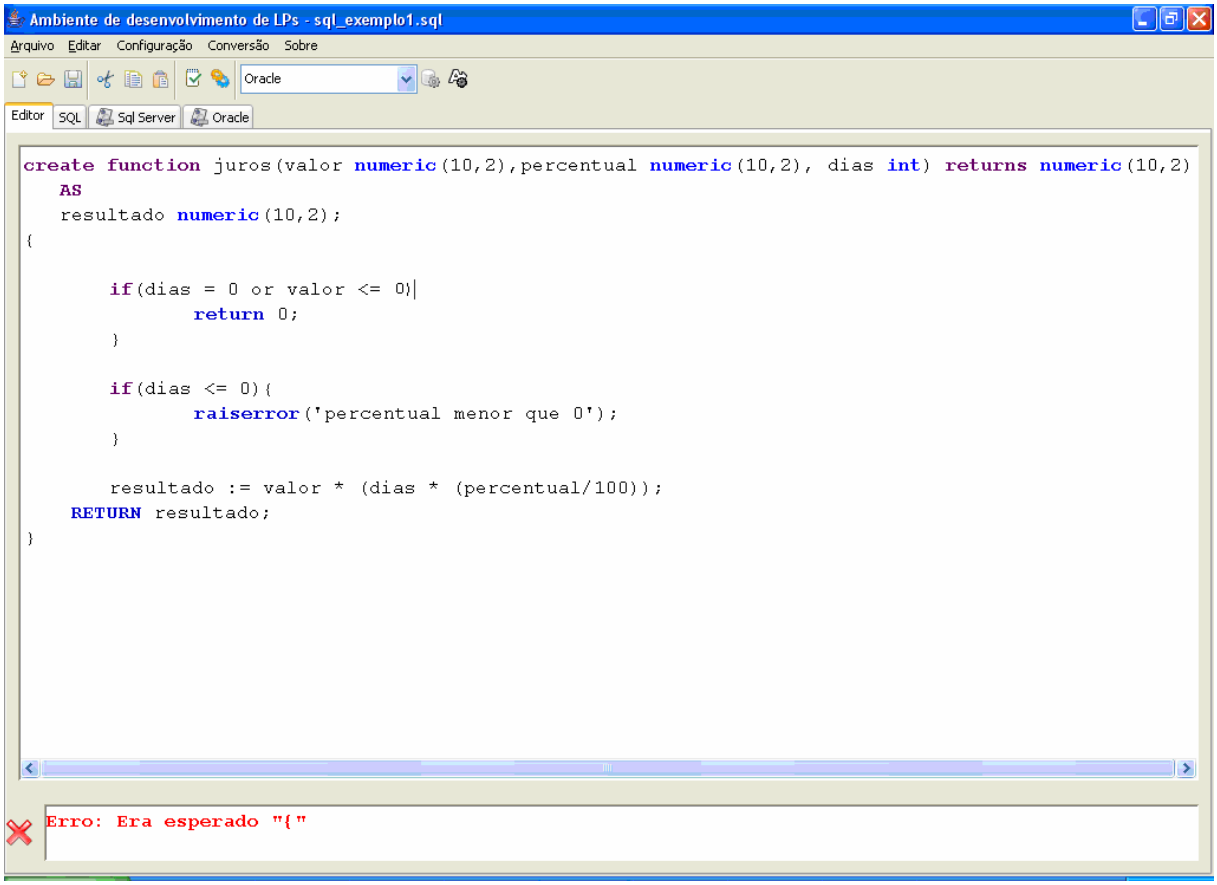


Figura 13 - Validação do código da LP

Na Figura 13 é possível visualizar a funcionalidade de sintaxe *highlight*, que destaca as cores das palavras reservadas da LP, conforme o seu agrupamento como pode ser observado nos tipos de variáveis “*numeric*” e “*int*”.

A Figura 14 demonstra o funcionamento do auto completar, que é exibido quando é digitada ao menos uma letra da palavra que se deseja completar ou visualizar as opções e pressionado <Ctrl> e <Espaço>, caso não encontrar nenhum ocorrência a caixa de opções não será exibida. A seleção da palavra pode ser feita a partir de um duplo clique do mouse ou movimentando as setas do teclado até a opção e pressionando a tecla <Enter>, para esconder a caixa basta inserir um espaço ou clicar com mouse sobre o editor.

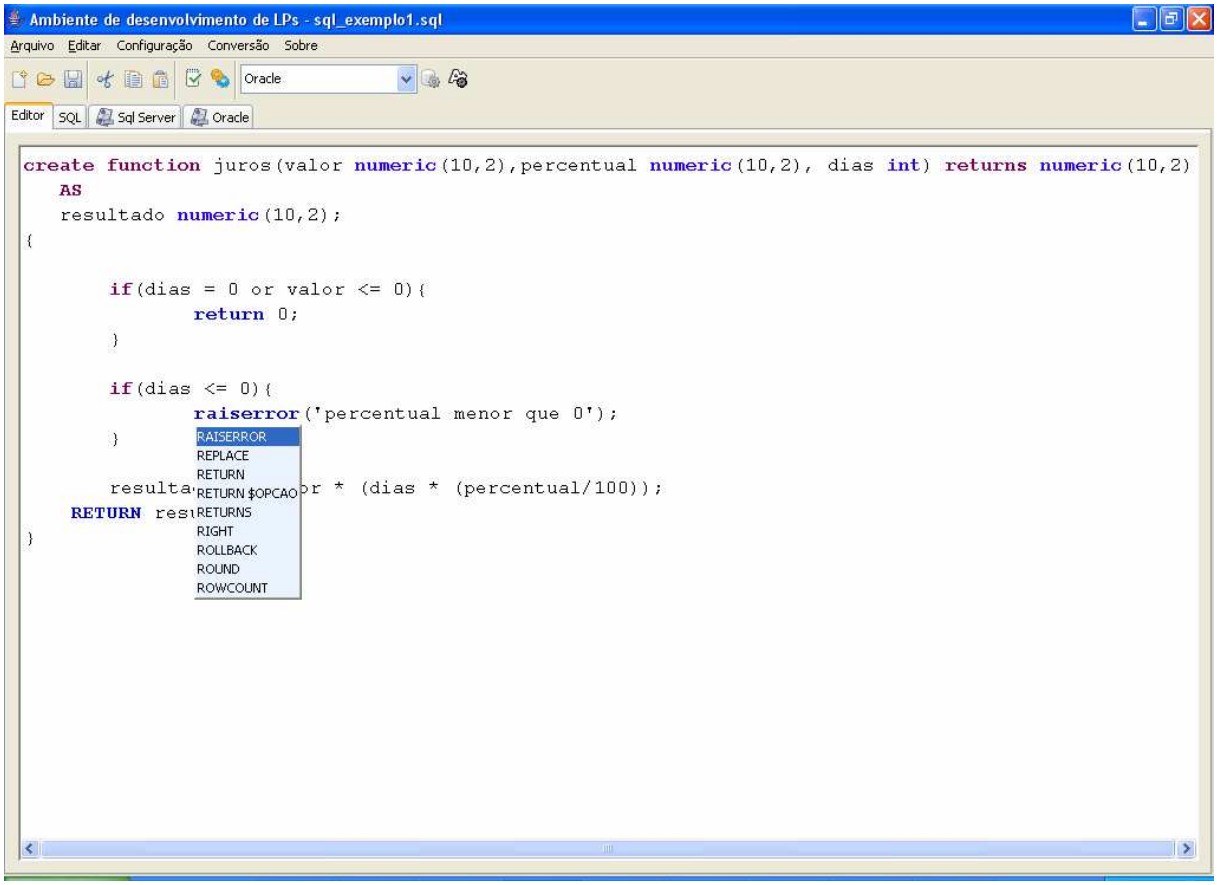


Figura 14 - Auto complemento

A Figura 15 mostra a tela de configuração de conexões que podem ser utilizadas para execução de *script* ou consultas nos SGBDs. A partir da tela pode ser configurado qualquer banco de dados que utilize *driver* JDBC.

Figura 15 - Tela de conexão com o banco de dados

Os campos obrigatórios da tela são “JDBC Driver” e “JDBC URL”. Usuário e senha não são necessários informar, pois a conexão pode ser efetuada informando o usuário e senha diretamente no campo “JDBC URL”. Para mais informações sobre como preencher os campos, cada um dos campos possui um botão de ajuda para auxiliar o desenvolvedor como no exemplo da Figura 16 que demonstras a ajuda do campo “JDBC URL”.

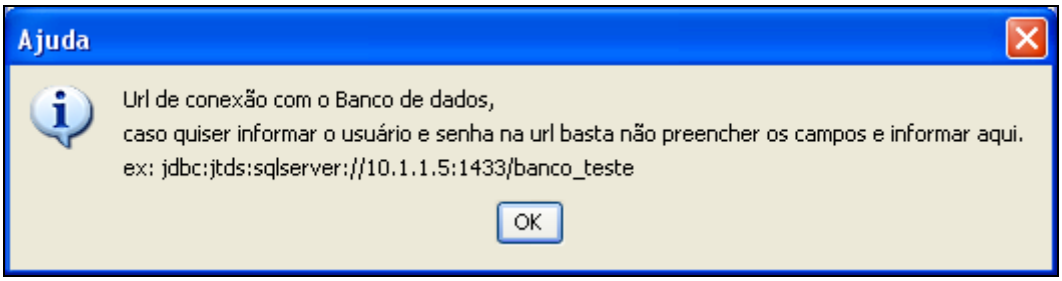


Figura 16 - Ajuda do campo *Driver url*

Os arquivos com extensão .jar que contém os *drivers* JDBC devem ser copiados para a pasta “JDBC” que localiza na raiz do sistema, conforme a mensagem na tela, para quando o sistema for iniciado carregue os *drivers*.

Após o preenchimento dos campos é possível salvar a conexão, apagar e também efetuar testes de conexão que caso obtiver sucesso, será exibida uma mensagem de confirmação.

Na Figura 17 é apresentada a tela para execução de consultas SQL utilizando com comando `SELECT`, onde basta ao desenvolvedor informar o código na área e clicar sobre o botão executar. O sistema irá executar a consulta utilizando a conexão selecionada e exibir o conteúdo em uma tabela. No momento da execução o ambiente efetua a conexão com o SGBD e executa a consulta caso ocorrer algum erro durante este processo será exibida uma mensagem como a da Figura 18, com a mensagem de erro.

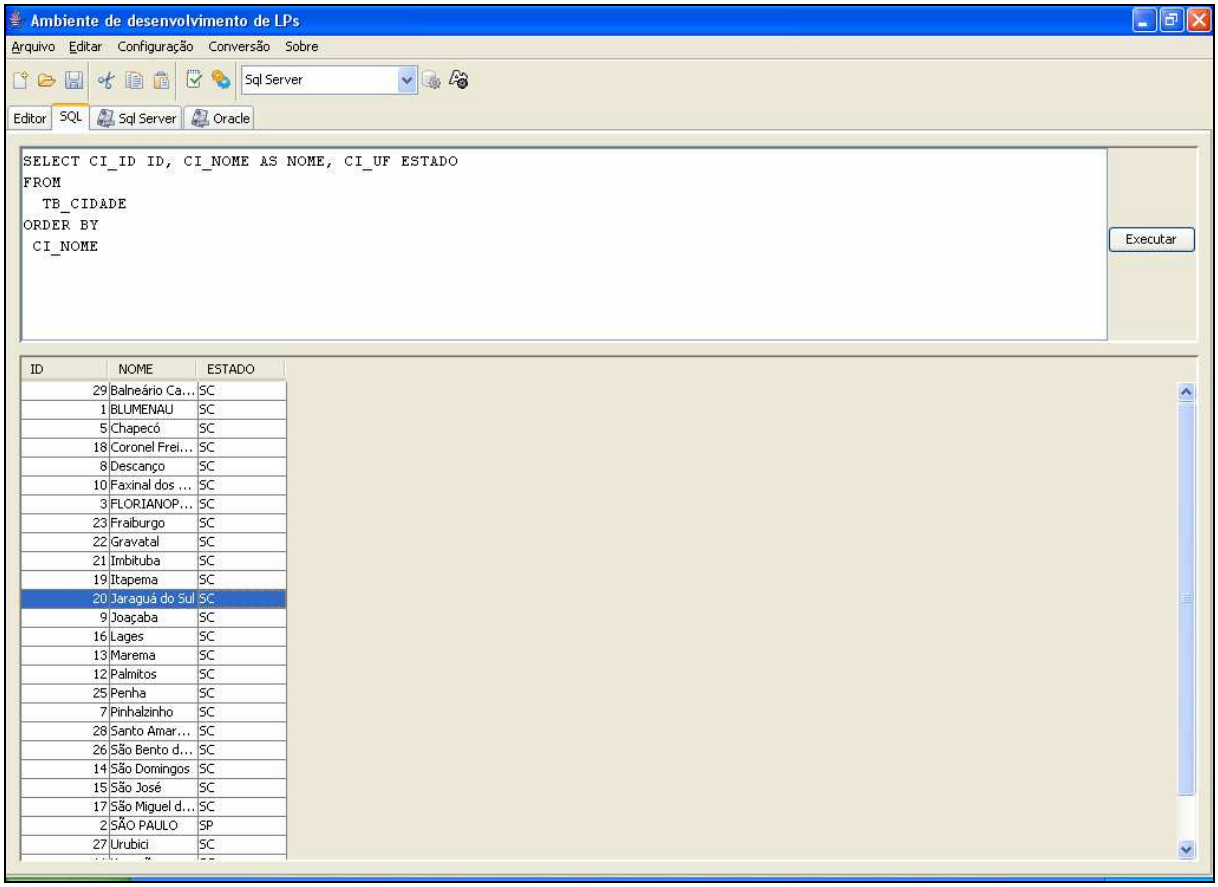


Figura 17 - Consultas SQL

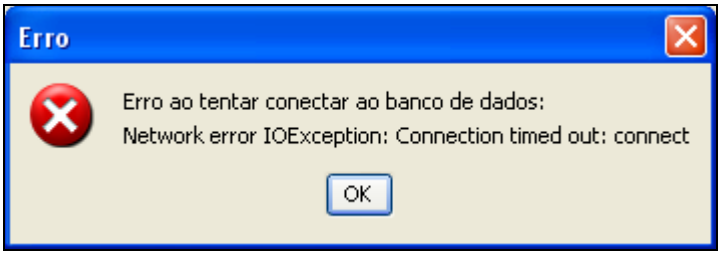


Figura 18 - Erro ao conectar ao banco de dados

Na Figura 19 é apresentada à tela de manutenção de *templates*, que permite a configuração dos mesmos, bem como a importação e exportação. A tela possui dois campos

na parte superior para informar o nome do *template* e outro para informar se o *template* está ativo, a funcionalidade de ativo serve para informar se o *template* será adicionado nas abas do editor e se será efetuada a geração de código.

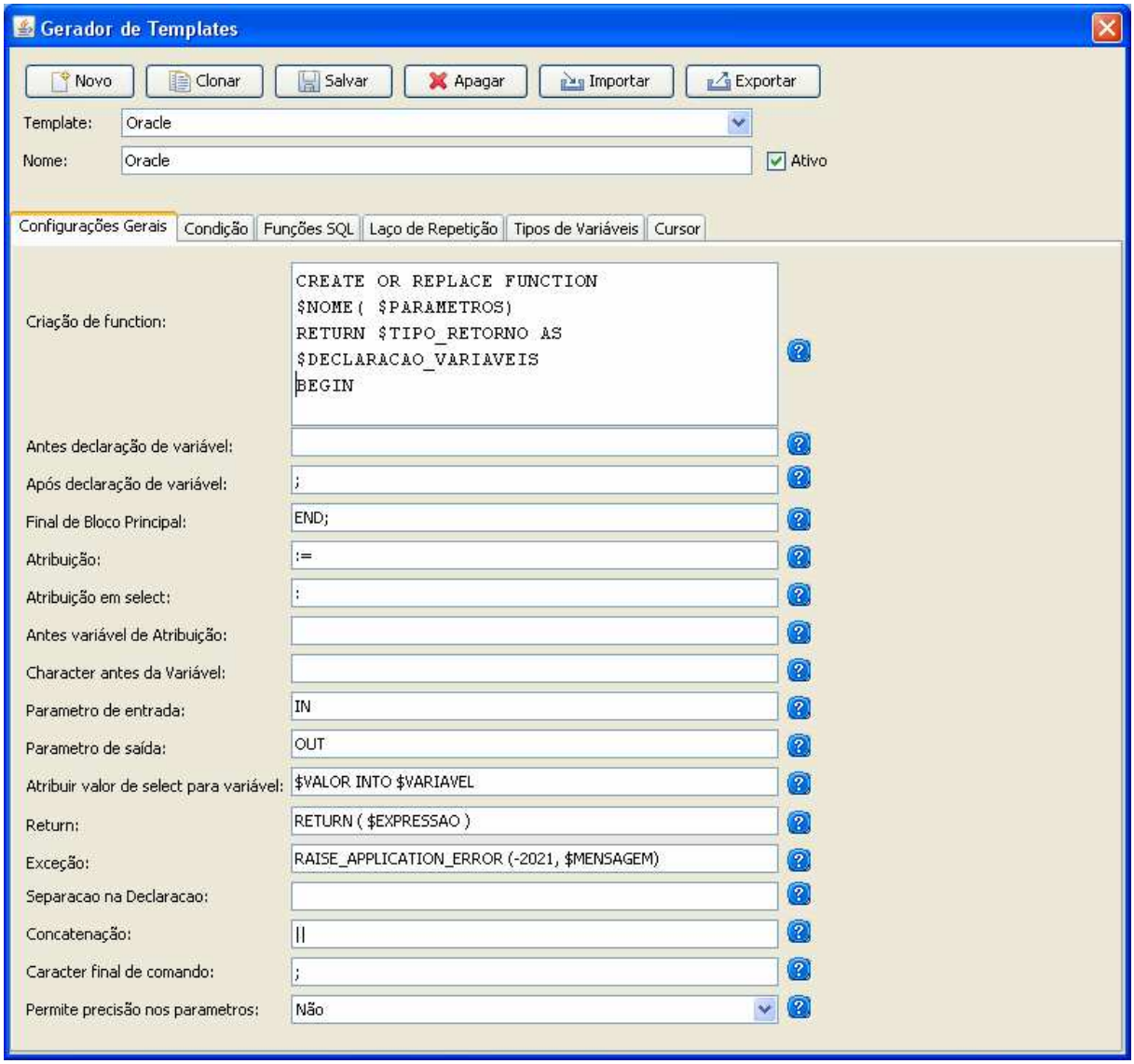


Figura 19 - Manutenção de *templates*

Na tela existem várias abas que contém grupamentos conforme as características das funcionalidades que poderão ser customizadas conforme o *template* da LP. Todos os campos possuem botão de ajuda que contendo informações explicativas sobre o preenchimento do campo, como demonstrado na Figura 20. Na Figura 20, podem ser observadas palavras que iniciam com $, estes são campos coringas, que podem ser ou não utilizados em qualquer posição do campo e serão substituído pelo valor que está descrido ao lado. Na Figura 19, o campo $NOME que irá substituir o nome da *function*. Nos campos que não possuírem campos com variáveis será efetuada uma simples substituição de valores, como é o exemplo do “Parâmetro de entrada” que será simplesmente substituído por ”IN”. As características que

não são contempladas pela LP podem ser deixadas em branco, o sistema obriga somente informa o nome do *template*.

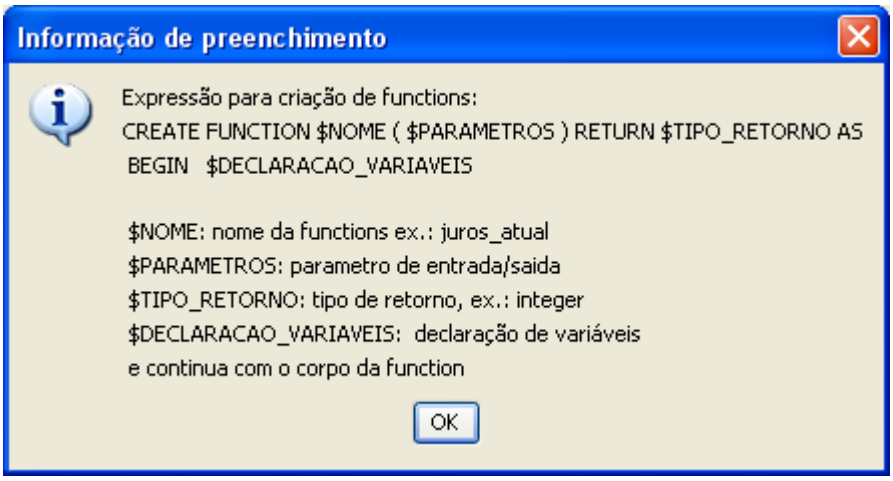


Figura 20 - Ajuda do campo *Create*

O botão “Clonar” efetua uma cópia fiel do *template* selecionado e concatena ao final do nome a palavra “(Clone)”. Esta funcionalidade é de muita utilidade para LPs que tenham sintaxes similares.

O botão “Importar”, possibilita a importação de um *template*, já definido pelo sistema ou que seja customizado via XML. No caso o arquivo seja inválido o sistema exibe uma mensagem de erro conforme a Figura 21. Quando o usuário for importar um *template* com o um nome que já existe, o sistema questionará o desenvolvedor se deseja substituir, e caso haja uma resposta negativa, o sistema importa o *template*, concatenando a data e hora no final do nome.

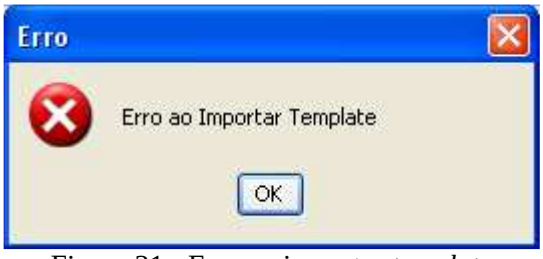


Figura 21 - Erro ao importar *template*

O botão “Exportar” efetua a exportação de *templates*, para arquivos XML solicitando o local que será salvo o mesmo. O Apêndice B contém informações sobre a estrutura dos *templates* exportados.

Os botões “Salvar” e “Apagar” como o nome já descreve, salva o *template* exibindo mensagem de confirmação e exclui o *template* solicitando confirmação.

O Quadro 17 e o Quadro 18 apresentam a geração de código para a SQL Server 2005 e Oracle, respectivamente, da *function* do Quadro 16. A geração foi feita utilizando os

templates disponíveis no ambiente. A partir de um *template* selecionado é possível efetuar a execução da LP no banco de dados, onde se ocorrer tudo certo será mostrada uma mensagem de “Comando executado com sucesso”.

```
CREATE FUNCTION NOMETESTE(NOME VARCHAR(255),SOBRENOME VARCHAR(255),
ANONASC INT) RETURNS VARCHAR(255)
   AS
   NOMECOMPLETO VARCHAR(255);
   TAMANHO VARCHAR(255);
   FRASE VARCHAR(255);
   DIFERENCA INT;
   DATAATUAL DATE;
{
      IF(LENGTH(NOME ||SOBRENOME ) > 0){
      NOMECOMPLETO := 'Nome completo: '||NOME||''||SOBRENOME||' | ';
      NOMECOMPLETO := NOMECOMPLETO||' Nome Maiúsculo: '||UPPER( nome )||' |
Nome Minúsculo: '||LOWER( SOBRENOME );
      NOMECOMPLETO := NOMECOMPLETO||''| As 3 primeiras letras: '|| SUBSTR(
NOME ||' '|| SOBRENOME, 1, 3  );
      TAMANHO :=  LENGTH( NOME ||' '|| SOBRENOME );
      NOMECOMPLETO := NOMECOMPLETO||''| A quantidade de letras do nome e
sobrenome é: '||  TAMANHO;
      FRASE := 'O ano de nascimento é $ano tem aproximadamente $diferenca';
      DATAATUAL := GETDATETIME;
      DIFERENCA := YEAR( DATAATUAL ) - ANONASC;
      FRASE := REPLACE ( FRASE , '$ano' , ANONASC );
      FRASE := REPLACE ( FRASE , '$diferenca' , DIFERENCA );

      NOMECOMPLETO := NOMECOMPLETO||' | '||FRASE;
      }ELSE{
            NOMECOMPLETO:= 'Nome não informado';
      }

    RETURN NOMECOMPLETO;
}
```

Quadro 16 - Função exemplo desenvolvido no ambiente

```
CREATE FUNCTION NOMETESTE  (@NOME VARCHAR(255), @SOBRENOME VARCHAR(255),
@ANONASC INT) RETURNS VARCHAR(255) AS BEGIN
   DECLARE @NOMECOMPLETO VARCHAR( 255 );
   DECLARE @TAMANHO VARCHAR( 255 );
   DECLARE @FRASE VARCHAR( 255 );
   DECLARE @DIFERENCA INT;
   DECLARE @DATAATUAL DATETIME;
   IF (LEN ( @NOME +@SOBRENOME  ) > 0  )
   BEGIN
      SET @NOMECOMPLETO = 'Nome completo: ' +@NOME +'' +@SOBRENOME +' | '
;
      SET  @NOMECOMPLETO  =  @NOMECOMPLETO  +'  Nome  Maiúsculo:  '  +UPPER  (
@NOME  ) +' | Nome Minúsculo: ' +LOWER ( @SOBRENOME  )  ;
      SET @NOMECOMPLETO = @NOMECOMPLETO +'| As 3 primeiras letras: '
+SUBSTRING ( @NOME +' ' +@SOBRENOME  , 1  , 3  )  ;
      SET @TAMANHO = LEN ( @NOME +' ' +@SOBRENOME  ) ;
      SET @NOMECOMPLETO = @NOMECOMPLETO +'| A quantidade de letras do nome
e sobrenome é: ' +@TAMANHO ;
      SET @FRASE = 'O ano de nascimento é $ano tem aproximadamente
$diferenca' ;
      SET @DATAATUAL = GETDATE();
      SET @DIFERENCA = YEAR ( @DATAATUAL )  - @ANONASC ;
      SET @FRASE = REPLACE ( @FRASE  , '$ano'  , @ANONASC  ) ;
      SET @FRASE = REPLACE ( @FRASE  , '$diferenca'  , @DIFERENCA  ) ;
      SET @NOMECOMPLETO = @NOMECOMPLETO +' | ' +@FRASE ;
   END
      ELSE
      BEGIN
         SET @NOMECOMPLETO = 'Nome não informado' ;
      END
   RETURN @NOMECOMPLETO  ;
END
```

Quadro 17 - Função gerada para SQL Server

```
CREATE OR REPLACE FUNCTION NOMETESTE ( NOME VARCHAR, SOBRENOME VARCHAR,
ANONASC INT) RETURN VARCHAR AS
    NOMECOMPLETO VARCHAR( 255 );
    TAMANHO VARCHAR( 255 );
    FRASE VARCHAR( 255 );
    DIFERENCA INT;
    DATAATUAL DATE;
BEGIN
   IF( LENGTH( NOME ||SOBRENOME  ) > 0  ) THEN
       NOMECOMPLETO := 'Nome completo: ' ||NOME ||'' ||SOBRENOME ||' | ' ;
       NOMECOMPLETO := NOMECOMPLETO ||' Nome Maiúsculo: ' ||UPPER( NOME  )
||' | Nome Minúsculo: ' ||LOWER( SOBRENOME ) ;
       NOMECOMPLETO := NOMECOMPLETO ||'| As 3 primeiras letras: '
||SUBSTR( NOME ||' ' ||SOBRENOME , 1 , 3   ) ;
       TAMANHO := LENGTH( NOME ||' ' ||SOBRENOME  ) ;
       NOMECOMPLETO := NOMECOMPLETO ||'| A quantidade de letras do nome e
sobrenome é: ' ||TAMANHO ;
       FRASE := 'O ano de nascimento é $ano tem aproximadamente
$diferenca' ;
       DATAATUAL := SYSDATE;
       DIFERENCA := TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'),9999) - ANONASC ;
       FRASE := REPLACE( FRASE , '$ano' , ANONASC ) ;
       FRASE := REPLACE( FRASE , '$diferenca' , DIFERENCA ) ;
       NOMECOMPLETO := NOMECOMPLETO ||' | ' ||FRASE ;
    ELSE
       NOMECOMPLETO := 'Nome não informado' ;
    END IF;
   RETURN ( NOMECOMPLETO  ) ;
END;
```

Quadro 18 - Função gerada para Oracle

3.5 RESULTADOS E DISCUSSÃO

O ambiente desenvolvido mostrou útil para a utilização em conversões de LPs, que elimina a necessidade de conhecer outras LPs. O editor com sintaxe *highlight* e auto completar de código melhoram a visibilidade e auxiliam nas dúvidas de sintaxes de comando, nome de tabelas e colunas.

A utilização de *templates* foi uma boa escolha como método de geração de código. Os *templates* não limitaram o ambiente à utilização de geração de linguagens já definidas, mas possibilitam aos desenvolvedores programar novas linguagens e também ajustar as disponibilizadas no ambiente conforme suas necessidades.

A conexão com diferentes SGBDs foram possíveis através do uso da API JDBC e os diversos *drivers* de cada banco, pois todos possuem métodos de conexões padronizados. No ambiente foi utilizado o tipo quatro (Protocolo-Nativo com *driver* Java).

A definição da linguagem gerou grandes dificuldades, levando em consideração a necessidade de criar uma linguagem com características de ser mais genérica possível, de fácil uso e que contemplasse as principais características de uma LP. Assim foi criada uma LP baseada na sintaxe do PL/SQL e T-SQL e com algumas características do Java como “{“ e “}” para definição de blocos.

A ferramenta GALS, mostrou-se muito útil auxiliando a definição e testes da linguagem. Na geração dos analisadores léxicos e sintáticos e as ações semânticas foi de extrema utilidade, pois facilitou muito a detecção de erro e conversão da linguagem.

Em relação aos trabalhos correlatos foi criado um comparativo entre o trabalho proposto e o desenvolvido por Bianchi (2007), conforme o Quadro 19. Os trabalhos possuem linha de pesquisa similar, com objetivo de gerar objetos de banco de dados a partir de linguagens padrões.

CARACTERÍSTICAS	Bianchi	Este trabalho
ferramenta utilizada para geração de analisadores léxicos e semânticos	GALS	GALS
linguagem utilizada para desenvolvimento	Delphi	Java
Objeto de banco de dados a ser gerado	*stored procedure*	*functions*
banco de dados que são geradas as LP	Oracle, SQL Server e PostgreSQL	Oracle e SQL Server
executa as LPs em banco de dados	sim	sim
uso de *templates* para geração de código	não	sim
permite o usuário alterar a geração de código	não	sim
permite conexão com vários SGDBs	não	sim
utilização do motor de *templates* Velocity	não	sim
executa consultas `SELECT` no banco de dados	não	sim
possui funcionalidades que auxiliam no desenvolvimento do código	não	sim

Quadro 19 - Comparativo entre o trabalho proposto e o de Bianchi (2006)

No trabalho foi descrito as principais características de um ambiente de desenvolvimento. As características implementadas no ambiente desenvolvido foram:

a) editor de código fonte: foi implementado e adicionado as funcionalidade de auto complete e sintaxe *highlight*;

b) compilador: implementado com as funcionalidade de analisadores léxicos, sintático e semântico, geração de código e verificação de erros, com auxilio para encontrar inconsistências;
c) execução: implementado funcionalidade para executar os resultados da geração de código.

4 CONCLUSÕES

O trabalho apresentou uma forma viável e ágil para a criação de *functions* de SGBDs, pois auxilia o desenvolvedor durante o processo de criação, disponibilizando funcionalidades como auto completar e sintaxe *highlight* e proporciona maior agilidade ao possibilitar escrever o código somente uma vez e a partir disso converter para os SGBDs definidos nos *templates*.

A utilização de *templates* para a geração de código entre as LPs permitiu que o código gerado não ficasse dependente do ambiente e as LPs utilizadas não se limitassem às disponibilizadas pelo ambiente. A codificação dos *templates* fica a critério do desenvolver que customiza conforme suas necessidades e características da LP.

Nos *templates* é utilizado a API Velocity para dar maior flexibilidade na customização. A implementação de *templates* não requer conhecimento em Velocity, pois todas as instruções necessárias para o preenchimento dos campos são encontradas em botões de ajuda ao lado dos campos.

Os pontos negativos da definição de uma linguagem genérica, é que impossibilita a utilização de recursos específicos das LPs. A utilização de *templates* para geração faz com que as conversões da linguagem genérica sejam limitadas às funcionalidades definidas nos *templates* e que o código gerado seja padronizado, mas não necessariamente o mais adequado.

A APIs `xstream-1.0.1.jar e xpp3-1.1.3.3_min.jar,` que foram utilizadas para serialização de objetos para formato XML, são muito úteis e de fácil utilização. As características que podem ser destacadas são: a de gerar código XML de fácil entendimento, serializar objetos suas dependências, possibilitar criar apelidos para os nomes das classes e extinguir a necessidade de criar rotinas para importar e exportar objetos.

A geração de código para as linguagens PL/SQL e T-SQL mostrou-se satisfatória, convertendo as funcionalidades básicas de uma LP levando em consideração as características de cada banco de dados.

4.1 EXTENSÕES

Sugere-se as seguintes extensões:

a) definir novas funcionalidades para a o ambiente, como por exemplo: busca e substituição, marcadores, perfis, integração com algum controle de versão;
b) modelar a estrutura da linguagem a fim de permitir a geração de *trigger* e *view* e incluir novas funcionalidades e novas estruturas na LP;
c) criar *wizards* que auxiliem na criação de *functions*;
d) criar uma perspectiva que contenha um depurador para LP;
e) implementar comandos específicos para cada LP;
f) efetuar a conversão de sintaxes com estruturas diferentes como, `rownum` ou `top` que limitam a quantidade de linhas de um comando `SELECT;`
g) desenvolver um controle de projetos.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Velocity**. [S.l.], 2004. Disponível em: <http://jakarta.aapache.org/velocity>. Acesso em: 07 set. 2007.

BEZERRA, E. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2002.

BIANCHI, Davi R. **Ferramenta para desenvolvimento de objetos de banco de dados em linguagem procedural**. 2006. 52 f. Trabalho de Conclusão do Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

DEITEL, Harvey M.; DEITEL, Paul J. **Java**: como programar. 6. ed. Porto Alegre: Pearson, 2005.

FANDERUFF, Damaris. **Oracle 8i**: utilizando SQL, PLUS e PL-SQL. São Paulo: Makron Books, 2000.

KLUG, Maicon. **Gerador de código JSP baseado em projeto de banco de dados**. 2007. 121 f. Trabalho de Conclusão do Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

HERRINGTON, Jack. **Code generation in action**. Greenwich: Manning Puplications, 2003.

HEUSER, Carlos A. **Projeto de banco de dados.** 3. ed. Porto Alegre: Sagra Luzzatto, 2000.

HIEBERT, Denis. **Protótipo de um compilador para a linguagem PL/SQL**. 2003. 52 f. Trabalho de Conclusão do Curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

MOURA, Maria F.; CRUZ, Sergio A. B. **Formatação de dados usando a ferramenta Velocity**. Campinas, 2002. Disponível em: <http://www.cnptia.embrapa.br/modules/tinycontent3/index.php?id=2>. Acesso em: 19 maio 2007.

O´HEARN, Steve. **OCP developer PL/SQL program units exam guide.** Berkeley, Calif: McGraw-Hill / Osborne, 2002.

PRICE, Ana. M. A., TOSCANI, Simão. **Implementação de Linguagens de Programação**: compiladores, 2. Ed. Porto Alegre: Sagra Luzzatto, 2001.

POSTGRESSSQL. **Tradução da documentação do PostgreSQL para o português do Brasil**. [S.I.], [2006?]. Disponível em: <http://pgdocptbr.sourceforge.net/pg80/plpgsql-porting.html>. Acesso em: 1 maio 2007.

REZENDE, Ricardo. **PL/SQL Developer**. [S.I], [2004?]. Disponível em: <http://www.sqlmagazine.com.br/revista.asp>. Acesso em: 07 out. 2007.

SEVERO, Carlos E. P. **NetBeans IDE 4.1**: para desenvolvedores que utilizam a tecnologia Java. Rio de Janeiro: Brasport, 2005.

TAKECIAN, Pedro L. et al. **Sistema de gerenciamento de workflows**. 2004. 35 f. Trabalho de Formatura Supervisionado (Bacharelado em Ciência da Computação) – Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo. Disponível em: <http://gul.linux.ime.usp.br/~plt/mac499/monografia.pdf >. Acesso em: 19 out. 2007.

URMAN, Scott. **Oracle 9i**: programação PL/SQL. Rio de Janeiro: Campus, 2002.

VALDAMERI, Alexander R. **AVR**. [S.I], [2003?]. Disponível em: <http://www.inf.furb.br/~arv/ava/fontes/gramatica2006-1.gals>. Acesso em: 01 ago. 2007.

APÊNDICE A – Gramática da linguagem procedural

O reconhecimento dos *token*, da linguagem foram implementados utilizando expressões regulares na ferramenta GALS, conforme as definições do Quadro 20:

```
ident =  (letra) (  (letra) | (digito)  )*
integer = (digito) ((digito) )*
float = (digito) (digito)* ("."(digito) (digito)*)?
Data = "'" (digito) (digito) "-" ((digito) (digito) | (letra) (letra)
(letra)) "-" (digito) (digito) ((digito) (digito))? "'"
SQLString  = "'" (qualquer letra que seja aspas)* "'"
```

Quadro 20 - Expressões regulares

As regras para construção de LPs no ambiente estão descritas no quando Quadro 21, utilizando a notação BNF.

```
<inicio>::= <create_alter> <function_procedure>;
<function_procedure>::= <function_create> | <procedure_create> ;
<function_create>::= FUNCTION  ident  "("  <paramentros>  ")"  RETURNS  <DataType_parametro>
<beginComandos>;
<procedure_create>::= PROCEDURE  ident  "("  <paramentros>  ")"  <beginComandos>;
<create_alter> ::= CREATE  | ALTER ;
<paramentros> ::= ident <tipoParamentro> <DataType_parametro> <paramentro_opc>|î;
<paramentro_opc>::= "," <paramentros>|î;
<tipoParamentro> ::= IN  <tipoParamentro_opc> | OUT  |î;
<tipoParamentro_opc> ::= OUT  | î;
<beginComandos> ::= AS <declaracao_variaveis> "{"  <SQLTest> "}";
<SQLTest> ::=  <Comando>  <PontoVirgula> <SQLTest_opc>;
<SQLTest_opc>::= <SQLTest>  | î;
<Comando> ::= <SelectSQL_into> ";"
              | <UpdateSQL> ";"
            | <InsertSQL> ";"
              | <DeleteSQL> ";"
              | <Transaction>
            | <atribuicao>";"
            | <condicaoIF>
            | <loop>
            | <while>
            | <for>
            | <cursor>";"
            | <return> ";"
            | <exeption>";" ;
<declaracao_variaveis> ::= <declaracao_OPC> | î ;
<declaracao_OPC>::= ident <Constant> <DataType> <Inicializacao_var>";"
<declaracao_variaveis>|
                   CURSOR  ident  IS <SelectSQL> ";" <declaracao_variaveis>;
<Inicializacao_var> ::=  ":=" <Field> | î ;
<Constant>::= CONSTANT | î ;
<condicaoIF> ::= IF "("  <loop_condiction>  ")" "{"7 <SQLTest> "}" <ELSEIF>;
<ELSEIF>::= ELSE  <ELSEIF_Opc> | î ;
<ELSEIF_Opc>::= "{" <SQLTest> "}" | <condicaoIF> ;
<loop> ::= LOOP  <SQLTest> EXIT WHEN "("  <loop_condiction>  ")" END LOOP;
<while> ::= WHILE "("  <loop_condiction> ")" LOOP4 <SQLTest> END  LOOP;
<for> ::= FOR <Field> IN integer_".."integer_  LOOP <SQLTest>  END LOOP  ;
<loop_condiction>::= <SimpleExpression_cond> <RelationSimpleExp_cond>;
<SimpleExpression_cond> ::= <NotOpcional> <Term_cond> <RepeteOperator_cond>;
<Term_cond> ::= <MenosOpcional> <K_cond>;
<K_cond> ::= <FieldTest_cond> | <ColumnFunction> | <FunctionExpr> | "(" <ExpSubSel> ")" ;
<FieldTest_cond> ::= <Field_cond> <TestOpcional>;
<Field_cond> ::=  ident <Field_cond_opc> | NULL  | float_  | integer_  | SQLString  | Data
|<Param>;
<Field_cond_opc>::= "%"<CURSOR_ATRIBUTOS>| î ;
<atribuicao>::=   ident ":=" <atribuicao_OPC> ;
<atribuicao_OPC> ::=  NULL
                    | ident <atribuicao_OPC_repet>
                    | <MenosOpcional> <tipo_Numerico> <atribuicao_OPC_repet>
                    | <funcoes>  <atribuicao_OPC_repet>
```

```
                    | Data <atribuicao_OPC_repet>
                    | SQLString <atribuicao_OPC_repet>  ;
<atribuicao_OPC_repet> ::= <juncao> <atribuicao_OPC_repet_1> | î ;
<juncao>::=<MathOperator>  | "||";
<atribuicao_OPC_repet_1>::= <atribuicao_OPC> | "(" <atribuicao_OPC> ")";
<cursor>::= OPEN  ident  | FETCH  ident    INTO <FieldList> | CLOSE  ident  ;
<CURSOR_ATRIBUTOS>::= NOTFOUND |FOUND |ISOPEN |ROWCOUNT;
<funcoes>::= <funcoes_str> | <funcoes_data> | <funcoes_numericas> | <funcoes_outra>
           | <funcoes_usuario> | <funcoes_convesao>;
<funcoes_convesao>::= CAST "(" ident AS  <DataType_parametro> ")";
<funcoes_usuario>::= DBO"." ident "(" <funcoes_usuario_par> ")";
<funcoes_usuario_par>::= <funcoes> <funcoes_usuario_par_mais> | <Field>
<funcoes_usuario_par_mais> | î ;
<funcoes_usuario_par_mais>::= <MathOperator> <funcoes_usuario_par>
                            |"," <funcoes_usuario_par>
                            | î ;
<funcoes_outra>::=   <nvl_function>  | <COALESCE_function>;
<funcoes_numericas> ::=      <length_function> | <INSTR_function>
                     | <ROUND_function> | <TRUNC_function>
                     | <MOD_function>;
<funcoes_str> ::=  <upper_function>   | <lower_function>
                | <REPLACE_function> | <SUBSTR_function>;
<funcoes_data>::= <DAY_function>     | <MONTH_function>
                | <YEAR_function>    | <HOUR_function>
                | <MINUTE_function>  | <SECOND_function>
                | GETDATETIME;
<col_or_literal>::= ident <col_or_literal_opc>  | SQLString  <col_or_literal_opc>
                 | <funcoes_str> <col_or_literal_opc> | <funcoes_usuario>
<col_or_literal_opc>;
<col_or_literal_opc> ::= "||" <col_or_literal> | î ;
<upper_function> ::= UPPER "(" <col_or_literal>")";
<lower_function> ::= LOWER "("<col_or_literal>")";
<SUBSTR_function> ::= SUBSTR "("<col_or_literal>"," <numero_inteiro>  "," <numero_inteiro>
")";
<length_function> ::= LENGTH "("<col_or_literal>")";
<INSTR_function> ::= INSTR "("<col_or_literal>"," <col_or_literal> ")";
<REPLACE_function> ::= REPLACE "("<col_or_literal>","<col_or_literal> <REPLACE_func_opc> ")";
<REPLACE_func_opc>::= "," <col_or_literal> | î ;
<numero_inteiro> ::= <Menos_or_mais_Opcional> integer_;
<col_or_Integer> ::=   <Menos_or_mais_Opcional> <tipo_Numerico> <col_or_Integer_opc>
                    | ident <col_or_Integer_opc>
                    | <funcoes_numericas> <col_or_Integer_opc>
                    | <funcoes_usuario>  <col_or_Integer_opc>
                    | "(" <col_or_Integer> ")" <col_or_Integer_opc> ;
<col_or_Integer_opc>::= <MathOperator> <col_or_Integer> | î ;
<tipo_Numerico> ::= integer_ | float_ ;
<Menos_or_mais_Opcional>::= "+" | "-" | î ;
<nvl_function> ::= NVL "(" <col_or_literal>","<col_or_literal>  ")";
<COALESCE_function>::= COALESCE "(" <funcoes_usuario_par> ")";
<ROUND_function>::= ROUND "(" <col_or_Integer> <ROUND_function_opc> ")";
<ROUND_function_opc>::= "," <col_or_Integer>  | î ;
```

```
<TRUNC_function>::= TRUNC "(" <col_or_Integer> <TRUNC_function_opc>  ")";
<TRUNC_function_opc>::= "," <col_or_Integer>  | î ;
<MOD_function>::= MOD "("  <col_or_Integer> "," <col_or_Integer> ")";
<DAY_function>::= DAY "("  <col_or_date_string>  ")";
<MONTH_function>::= MONTH "(" <col_or_date_string> ")";
<YEAR_function>::= YEAR "(" <col_or_date_string> ")";
<HOUR_function>::= HOUR "(" <col_or_date_string>  ")";
<MINUTE_function>::= MINUTE "("  <col_or_date_string> ")";
<SECOND_function>::= SECOND "(" <col_or_date_string> ")";
<col_or_date_string>::=   ident | Data | <funcoes_usuario>;
<funcoes_select>::= <funcoes_str_select>   | <funcoes_data_select>
           | <funcoes_numericas_select>   | <funcoes_outra_select>
           | <funcoes_usuario_select>     | <funcoes_convesao_select>;
<funcoes_convesao_select>::= CAST "(" ident AS  <DataType_parametro_select> ")";
<funcoes_usuario_select>::= DBO"." ident "(" <funcoes_usuario_par_select> ")";
<funcoes_usuario_par_select>::= <funcoes_select> <funcoes_usuario_par_mais_select> | <Field>
<funcoes_usuario_par_mais_select> | î ;
<funcoes_usuario_par_mais_select>::= <MathOperator> <funcoes_usuario_par_select>
                           |"," <funcoes_usuario_par_select>
                           | î ;
<funcoes_outra_select>::=   <nvl_function_select> | <COALESCE_function_select>;
<funcoes_numericas_select> ::=      <length_function_select>  | <INSTR_function_select>
                     | <ROUND_function_select>          | <TRUNC_function_select>
                     | <MOD_function_select>;
<funcoes_str_select> ::=  <upper_function_select>  | <lower_function_select>
               | <REPLACE_function_select>       | <SUBSTR_function_select>;
<funcoes_data_select>::= <DAY_function_select>     | <MONTH_function_select>
               | <YEAR_function_select>              | <HOUR_function_select>
               | <MINUTE_function_select>        | <SECOND_function_select>
               | GETDATETIME;
<col_or_literal_select>::= ident <col_or_literal_opc_select>
                | SQLString  <col_or_literal_opc_select>
                | <funcoes_str_select> <col_or_literal_opc_select>
                | <funcoes_usuario_select> <col_or_literal_opc_select>;
<col_or_literal_opc_select> ::= "||" <col_or_literal_select> | î ;
<upper_function_select> ::= UPPER "(" <col_or_literal_select>")";
<lower_function_select> ::= LOWER "(<col_or_literal_select>")";
<SUBSTR_function_select> ::= SUBSTR "(<col_or_literal_select>"," <numero_inteiro_select>  ","
<numero_inteiro_select>  ")";
<length_function_select> ::= LENGTH "(<col_or_literal_select>")";
<INSTR_function_select> ::= INSTR "(<col_or_literal_select>"," <col_or_literal_select> ")";
<REPLACE_function_select> ::= REPLACE "(<col_or_literal_select>","<col_or_literal_select>
<REPLACE_func_opc_select> ")";
<REPLACE_func_opc_select>::= "," <col_or_literal_select> | î ;
<numero_inteiro_select> ::= <Menos_or_mais_Opcional_select> integer_;
<col_or_Integer_select> ::=   <Menos_or_mais_Opcional_select> <tipo_Numerico_select>
<col_or_Integer_opc_select>
                   | ident <col_or_Integer_opc_select>
                   | <funcoes_numericas_select> <col_or_Integer_opc_select>
                   | <funcoes_usuario_select>  <col_or_Integer_opc_select>
                   | "(" <col_or_Integer_select> ")" <col_or_Integer_opc_select> ;
```

```
<col_or_Integer_opc_select>::= <MathOperator> <col_or_Integer_select> | î ;
<tipo_Numerico_select> ::= integer_ | float_ ;
<Menos_or_mais_Opcional_select>::= "+" | "-" | î ;
<nvl_function_select> ::= NVL "(" <col_or_literal_select>","<col_or_literal_select>  ")";
<COALESCE_function_select>::= COALESCE "(" <funcoes_usuario_par_select> ")";
<ROUND_function_select>::= ROUND "(" <col_or_Integer_select> <ROUND_function_opc_select> ")";
<ROUND_function_opc_select>::= "," <col_or_Integer_select>  | î ;
<TRUNC_function_select>::= TRUNC "(" <col_or_Integer_select> <TRUNC_function_opc_select>  ")";
<TRUNC_function_opc_select>::= "," <col_or_Integer_select>  | î ;
<MOD_function_select>::= MOD "("  <col_or_Integer_select> "," <col_or_Integer_select> ")";
<DAY_function_select>::= DAY "("  <col_or_date_string_select>  ")";
<MONTH_function_select>::= MONTH "(" <col_or_date_string_select> ")";
<YEAR_function_select>::= YEAR "(" <col_or_date_string_select> ")";
<HOUR_function_select>::= HOUR "(" <col_or_date_string_select>  ")";
<MINUTE_function_select>::= MINUTE "("  <col_or_date_string_select> ")";
<SECOND_function_select>::= SECOND "(" <col_or_date_string_select> ")";
<col_or_date_string_select>::= ident | Data| <funcoes_usuario_select>;
<return> ::= RETURN  <atribuicao_OPC> ;
<exeption>::= RAISERROR "(" <col_or_literal>  ")";
<PontoVirgula> ::= ";" | î ;
<SelectSQL> ::=  <SelectStmt> <UnionSelect> ;
<SelectSQL_into> ::=  <SelectStmt_into>;
<UnionSelect> ::= UNION <AllOpcional> <SelectStmt> <UnionSelect> | î ;
<AllOpcional> ::= ALL | î ;
<SubSelectSQL> ::= <SelectSQL> ;
<UpdateSQL> ::= <UpdateStmt> ;
<InsertSQL> ::= <InsertStmt> ;
<DeleteSQL> ::= <DeleteStmt> ;
<UpdateStmt> ::= UPDATE <Table> SET <UpdateFieldList> <WhereOpcional> ;
<WhereOpcional> ::= <WhereClause> | î ;
<WhereOpcional_cursor> ::= <WhereClause_cursor> | î ;
<UpdateField> ::= <ColumnName> "=" <Field_values> ;
<UpdateFieldList> ::= <UpdateField> <UpdateRepete> ;
<UpdateRepete> ::= "," <UpdateField> <UpdateRepete>| î ;
<InsertStmt> ::= INSERT INTO  <Table> <ColumnListOpcional> <Values> ;
<Values> ::= <ValuesList> | <SelectSQL> ;
<ValuesList> ::= VALUES "(" <FieldList_values> ")";
<ColumnListOpcional> ::= "(" <ColumnList> ")" | î;
<DeleteStmt> ::= DELETE FROM <Table> <WhereOpcional>;
<SelectStmt> ::= <SelectClause>  <FromClause> <innerOptional> <WhereOpcional>
<GroupByClauseOpcional> <HavingClauseOpcional> <OrderByClauseOpcional> ;
<SelectStmt_into> ::= <SelectClause_into>  <FromClause>  <innerOptional>   <WhereOpcional>
<GroupByClauseOpcional> <HavingClauseOpcional> <OrderByClauseOpcional> ;
<innerOptional>::= <Inner_opcao> JOIN ident ON "(" <SearchCondition> ")" <innerOptional> |î;
<Inner_opcao>::= INNER | LEFT <Outer>;
<Outer>::= OUTER | î;
<OrderByClauseOpcional> ::= <OrderByClause> | î ;
<HavingClauseOpcional> ::= <HavingClause> | î ;
<GroupByClauseOpcional> ::= <GroupByClause> | î ;
<SelectClause> ::= SELECT <DistinctAllOpcional> <SelectFieldList>;
<SelectClause_into> ::= SELECT <DistinctAllOpcional> <SelectFieldList_into>;
```

```
<DistinctAllOpcional> ::= DISTINCT | ALL | î ;
<FromClause> ::= FROM <FromTableList>;
<FromTableList> ::= <QualifiedTable>;
<QualifiedSeparator> ::= "," <QualifiedTable> ;
<QualifiedTable> ::= ident <identOpcional>;
<identOpcional> ::=  "." ident <AsAliasOpcional>| î
                    | AS <Alias>              | ident ;
<AsAliasOpcional> ::=  AS <Alias> | ident | î ;
<WhereClause> ::=  WHERE   <SearchCondition> ;
<WhereClause_cursor> ::=  WHERE   <SearchCondition_cursor> ;
<HavingClause> ::=  HAVING   <SearchCondition> ;
<OrderByClause> ::=  ORDER BY   <OrderByFldList>;
<GroupByClause> ::=  GROUP BY   <FieldList>;
<SelectFieldList> ::= <SelectField> <SeparatorSelectField>;
<SelectFieldList_into> ::= <SelectField_into> <SeparatorSelectField_into>;
<SeparatorSelectField> ::= "," <SelectField> <SeparatorSelectField> | î ;
<SeparatorSelectField_into> ::= "," <SelectField_into> <SeparatorSelectField_into> | î ;
<SelectField> ::= <Expression> <AsAlias> | "*" ;
<SelectField_into> ::=  <Expression_into> INTO  ident ;
<AsAlias> ::= AS <Alias> |  î ;
<FunctionExpr> ::= <W> "(" <Expression> <SeparatorExpression> ")"
                | <funcoes>;
<FunctionExpr_select> ::= <W> "(" <Expression> <SeparatorExpression> ")"
                | <funcoes_select>;
<SeparatorExpression> ::= "," <Expression> <SeparatorExpression> |  î ;
<W> ::= TIMESTAMP;
<ColumnFunction> ::=  <Function> "(" <AsteriscoDistinct> ")";
<AsteriscoDistinct> ::= "*" | <DistinctExpression>;
<DistinctExpression> ::= <DistinctOpcional> <ColumnName_function> ;
<ColumnName_function> ::= ident <PontoCampo_function>;
<PontoCampo_function> ::= "." ident| î ;
<DistinctOpcional> ::= DISTINCT | î ;
<Function> ::= COUNT| SUM | MAX| MIN|AVG ;
<ColumnList> ::= <ColumnName>  <SeparatorColumnName>;
<SeparatorColumnName> ::= "," <ColumnName>  <SeparatorColumnName> | î;
<ColumnName> ::=  ident <PontoCampo>;
<PontoCampo> ::=  "." <IdentAsterisco> | "%" <CURSOR_ATRIBUTOS>| î ;
<IdentAsterisco> ::=  ident | "*"  ;
<FieldList> ::= <Field>  <SeparatorField> ;
<FieldList_values> ::= <Field_values>  <SeparatorField_values> ;
<SeparatorField> ::= "," <Field>  <SeparatorField> |  î ;
<SeparatorField_values> ::= "," <Field_values>  <SeparatorField_values> |  î ;
<Field> ::= <ColumnName> | NULL   | float_  | integer_ | SQLString  | Data |<Param>;
<Field_cursor> ::= <ColumnName> | NULL   | float_  | integer_ | SQLString  | Data
|<Param_cursor>;
<Field_values> ::= ident | NULL   | float_  | integer_ | SQLString  | Data |<Param>;
<Table> ::= ident ;
<Alias> ::= ident ;
<OrderByFldList> ::= <OrderByField> <SeparatorOrderBy>;
<SeparatorOrderBy> ::= "," <OrderByField> <SeparatorOrderBy> |  î ;
<OrderByField> ::= <ColumnInteger> <OrdemOpcional>;
```

```
<ColumnInteger> ::= <ColumnName> | integer_;
<OrdemOpcional> ::= DESC | ASC |  î ;
<SearchCondition> ::= <Expression>;
<SearchCondition_cursor> ::= <Expression_cursor>;
<Expression> ::= <SimpleExpression> <RelationSimpleExp>;
<Expression_into> ::= <SimpleExpression_into> <RelationSimpleExp>;
<Expression_cursor> ::= <SimpleExpression_cursor> <RelationSimpleExp_cursor>;
<RelationSimpleExp> ::= <Relation> <SimpleExpression> <RelationSimpleExp>  |  î ;
<RelationSimpleExp_cursor> ::= <Relation> <SimpleExpression_cursor> <RelationSimpleExp_cursor>
|  î ;
<RelationSimpleExp_cond> ::= <Relation> <SimpleExpression_cond> <RelationSimpleExp_cond>  |  î
;
<SimpleExpression> ::= <NotOpcional> <Term> <RepeteOperator>;
<SimpleExpression_cursor> ::= <NotOpcional> <Term_cursor> <RepeteOperator_cursor>;
<SimpleExpression_into> ::= <NotOpcional> <Term_into> <RepeteOperator_into>;
<RepeteOperator> ::=  <Operator> <NotOpcional> <Term> <RepeteOperator> |  î ;
<RepeteOperator_cond> ::=  <Operator> <NotOpcional> <Term_cond> <RepeteOperator_cond> |  î ;
<RepeteOperator_cursor> ::=  <Operator> <NotOpcional> <Term_cursor> <RepeteOperator_cursor> |
î ;
<RepeteOperator_into> ::=  <Operator> <NotOpcional> <Term_into> <RepeteOperator_into> |  î ;
<NotOpcional> ::= NOT  |  î ;
<Term> ::= <MenosOpcional> <K>;
<Term_cursor> ::= <MenosOpcional> <K_cursor>;
<Term_into> ::= <MenosOpcional> <K_into>;
<K> ::= <FieldTest> | <ColumnFunction> | <FunctionExpr_select> | "(" <ExpSubSel> ")" ;
<K_cursor> ::= <FieldTest_cursor> | <ColumnFunction> | <FunctionExpr> | "(" <ExpSubSel> ")" ;
<K_into> ::= <FieldTest> |  <ColumnFunction> | <FunctionExpr> | "(" <ExpSubSel> ")" ;
<ExpSubSel> ::= <Expression> | <SubSelectSQL>;
<FieldTest> ::= <Field> <TestOpcional>;
<FieldTest_cursor> ::= <Field_cursor> <TestOpcional>;
<TestOpcional> ::= <TestExpr> |  î ;
<MenosOpcional> ::= "-" | î ;
<Param> ::= ":" ident;
<Param_cursor> ::= ":"9 ident;
<Operator> ::= <MathOperator> | <WordOperator>;
<MathOperator> ::= "*" | "/" | "+" | "-" ;
<WordOperator> ::= AND | OR ;
<LikeTest> ::= LIKE <SqlParam> <EscapeOpcional> ;
<EscapeOpcional> ::= ESCAPE SQLString |  î ;
<SqlParam> ::= SQLString | <Param>;
<NullTest>::= IS <NotOpcional>  NULL  ;
<Relation> ::= "=" | "<>"| "<" | "<=" | ">" | ">=" ;
<TestExpr> ::= <NullTest> | <NotOpcional> <L> ;
<L> ::= <InSetExpr> | <BetweenExpr> | <LikeTest> ;
<BetweenExpr> ::= BETWEEN <Field> AND <Field>;
<InSetExpr> ::= IN "(" <FieldSub> ")";
<FieldSub> ::= <FieldList> | <SubSelectSQL>;
<Transaction> ::= <CommitRollBack> <WorkOpcional>;
<WorkOpcional> ::= WORK |  î ;
<CommitRollBack> ::= COMMIT | ROLLBACK ;
<lenParam> ::= "(" integer_ ")"  ;
```

```
<precision> ::= integer_  <IntOpcional> ;
<IntOpcional> ::= "," integer_  | î ;
<DataType> ::= <DataChar>| <DataVarChar>| <Int>  | SMALLINT | <DataNum>| DATE  | <dataDate>;
<DataType_parametro> ::= CHAR <precision_parametro_simple>
                      | CHARACTER<precision_parametro_simple>
                      | VARCHAR<precision_parametro_simple>
                      | VARCHAR2 <precision_parametro_simple>
                      | <Int>  | SMALLINT
                      | NUMERIC<precision_parametro_double>
                      | NUMBER <precision_parametro_double>
                      | DATE  |TIMESTAMP |  TIME;
<dataDate>::= TIMESTAMP <lenParam> | TIME <lenParam> ;
<precision_parametro_double>::= "("0 integer_0 <pricision_1> ")"0 ;
<pricision_1>::= ","0 integer_0 | î ;
<precision_parametro_simple>::= "("0 integer_0 ")"0 ;
<DataNum> ::=  NUMERIC  "(" <precision> ")"  | NUMBER  "(" <precision> ")"  ;
<Int> ::=  INTEGER | INT;
<DataVarChar> ::=  VARCHAR <lenParam> | VARCHAR2 <lenParam> ;
<DataChar> ::= <Char> <lenParam> ;
<Char> ::= CHAR | CHARACTER ;
```

Quadro 21 - BNF da LP do ambiente

APÊNDICE B – Templates de bancos de dados

Os *templates* gerados pelo sistema são baseados na serialização do objeto `Template` conforme mostrado na Figura 7. A seguir é apresentado um trecho de código do *template* do SQL Server este foi exportado utilizando o ambiente e pode ser observado que existem 10 *tags* as quais são:

a) `<banco>`: tag principal do arquivo xml;
b) `<ativo>`: informa se o *template* está ativo;
c) `<codigo>`: código do *template*;
d) `<nome>`: nome do *template*;
e) `<template>`: início da definição do *template*;
f) `<regras>`: inicia a lista de regras do *template*;
g) `<conversao>`: define a conversão de uma funcionalidade;
h) `<chave>`: palavra chave que identifica a funcionalidade;
i) `<observacao>`: informações de preenchimento do campo valor;
j) `<valor>`: instrução que será executada quando a funcionalidade for utilizada.

```
<banco>
  <ativo>true</ativo>
  <codigo>0</codigo>
  <nome>SQL Server</nome>
  <template>
    <regras>
      <conversao>
        <chave>varchar2</chave>
        <observacao> Expressão que deverá substituir 'VARCHAR2'Utilizado em: create function nome(in nome varchar) returns varchar(255) as declare    x VARCHAR2(10);{ ...  }</observacao>
        <valor>VARCHAR</valor>
      </conversao>
      <conversao>
        <chave>time</chave>
        <observacao> Expressão que deverá substituir 'TIME'Utilizado em: create function nome(in nome varchar) returns varchar(255) as declare     x TIME(10);{ ...  }</observacao>
        <valor>DATETIME</valor>
      </conversao>
      <conversao>
        <chave>numeric</chave>
        <observacao> Expressão que deverá substituir 'NUMERIC'Utilizado em: create function nome(in nome varchar) returns varchar(255) as declare    x NUMERIC(10); y NUMERIC(10,2);{    ...  }</observacao>
        <valor>NUMERIC</valor>
      </conversao>
      <conversao>
        <chave>separacaovardeclaracao</chave>
        <observacao> Expressão que deverá substituir ';' Utilizado para separar a lista de variáveis Utilizado em: create function nome(in nome varchar) returns varchar(255) as declare     nome varchar(255); &lt;-      idade integer;{    select dis_nome into nome from tb_disciplina;    IF(nome is null){       RAISERROR ( 'Não foi possivel ' || 'encontrar as disciplinas'   )   }   return nome;}</observacao>
        <valor></valor>
      </conversao>
      <conversao>
        <chave>createall</chave>
        <observacao>Expressão para criação de functions: CREATE FUNCTION $NOME ( $PARAMETROS ) RETURN $TIPO_RETORNO AS BEGIN   $DECLARACAO_VARIAVEIS  $NOME: nome da functions ex.: juros_atual $PARAMETROS: parametro de entrada/saida  $TIPO_RETORNO: tipo de retorno, ex.: integer  $DECLARACAO_VARIAVEIS:  declaração de variáveis  e continua com o corpo da function  </observacao>
        <valor>CREATE FUNCTION $NOME ($PARAMETROS) RETURNS $TIPO_RETORNO AS BEGIN $DECLARACAO_VARIAVEIS</valor>
```

```
      </conversao>
      <conversao>
        <chave>date</chave>
        <observacao> Expressão que deverá substituir 'DATE'Utilizado em: create function nome(in nome varchar) returns varchar(255) as declare    x DATE;{    ... }</observacao>
        <valor>DATETIME</valor>
      </conversao>
      <conversao>
        <chave>finalblocoelse</chave>
        <observacao> Expressão que deverá substituir '}'Utilizado em: create function nome(in nome varchar) returns varchar(255) as {    IF ( x = 1 ) then { x = 2;   }     ...  }</observacao>
        <valor>END</valor>
      </conversao>
      <conversao>
        <chave>parametroentrada</chave>
        <observacao> Expressão que deverá substituir 'IN'Utilizado em: create function nome(nome IN varchar) returns varchar(255) as {    x = x+1;       ... }</observacao>
        <valor>IN</valor>
      </conversao>
      <conversao>
        <chave>inicioblocoelse</chave>
        <observacao> Expressão que deverá substituir '{'Utilizado em: create function nome(in nome varchar) returns varchar(255) as {    IF ( x = 1 ) then { x = 2;   } ELSE {       x = 3;   }     ...  }</observacao>
        <valor>BEGIN</valor>
      </conversao>
...
</regras>
</template>

</banco>
```

Quadro 22 - *Template* do Oracle