Extensão da ferramenta Delphi2Java-II para suportar a tradução dos tratadores de eventos

Carlos Henrique Marian

Orientador: Mauro Marcelo Mattos

Blumenau, 10 de julho de 2008

Sumário

- Introdução
- Objetivos
- Conceitos
- Delphi2Java-II
- Trabalhos Correlatos
- Desenvolvimento
- Resultados
- Conclusão
- Extensões

Introdução

- Vantagens e Desvantagens do Delphi
- Mercado de desenvolvimento
- Plataformas em Destaque
- Por que migrar sistemas?
- Desafio da Migração de Software
- Delphi2Java-II

Objetivos

- Aprimorar ferramenta Delphi2Java-II
  - Identificar os eventos
  - Buscar o código fonte dos tratadores: leitura do arquivo .PAS
  - Analise do fonte Object Pascal com o arquivo Java gerado

Conceitos

- Geração de código
  - Compilador: léxico, sintático e semântico
- Arquivo DFM
- Classe que define o Formulário
- Tratadores de eventos em Delphi
- Tratadores de eventos em Java

Geradores de Código

- O que é geração de código?
  - Programas que geram programas
- Por que usar geração automática de código
  - Diminuição de custos e tempo
- Grupo de geradores de código
  - Ativo: produzem código automático
  - Passivo: produzem templates que devem ser preenchidos

Compilador

- Estrutura de um compilador
- Analisadores Léxico, Sintático e Semântico
  - Léxico: separa o fonte em símbolos/*tokens* que possuem significado para a linguagem
  - Sintático: verifica seqüencia de símbolos contidas no programa
  - Semântico: verifica incoerências nas construções utilizadas
- Ferramenta GALS

Arquivo DFM

- O que é um arquivo DFM
- Informações do DFM


```
01 object Form1: TForm1
02   Left = 389
03   Top = 244
04   Width = 191
05   Height = 102
06   Caption = 'Modelo em Delphi'
07   object Button1: TButton
08     Left = 56
09     Top = 24
10     Width = 75
11     Height = 25
12     Caption = 'Button 1'
13     OnClick = Button1Click
14   end
15 end
```

Classe que define o formulário

- Estrutura
- Informações

```
type
   TForm1 = class(TForm)
     Button1: TButton;
     procedure Button1Click(Sender: TObject);
   private
    { Private declarations }
   public
    { Public declarations }
  end;
var  Form1: TForm1;
```

Tratadores de eventos em Delphi

- O que são eventos?
  - Ações do usuário que a aplicação pode receber e tratar (CARVALHO, 1998, p. 18).
  - Exemplo: clique o usuário da sobre um botão.
- Estrutura do Tratador de eventos
  - Cada evento tratado possui um procedimento associado

```
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender:TObject);
begin
  ShowMessage('Mensagem ao Usuário.');
end;
end.
```

Tratadores de eventos em Java

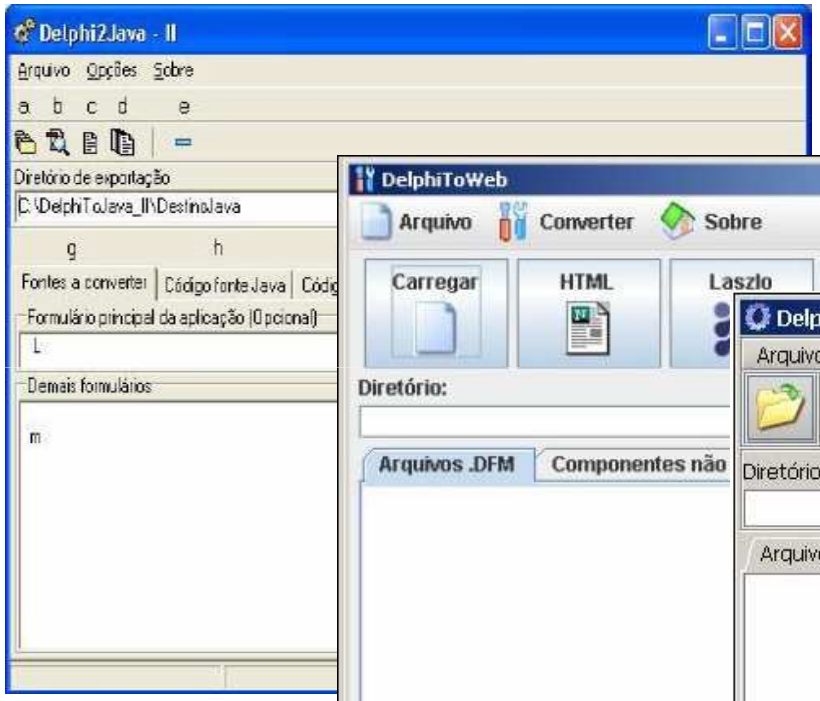
- Estrutura do Tratador de eventos
  - Pacotes `java.awt.event` e `java.swing.event`.
- *Listeners* – classes para tratar eventos e *Dispatch* – mecanismos de aviso.

```
// 1 - Criando um componente visual.
Button b1 = new Button(“Salvar");
// 2 - Criando um observador de Eventos através de uma classe anônima.
 ActionListener observadorDeB1 = new java.awt.event.ActionListener() {
 // 2.1 - implementando o método para receber uma notificação.
 public void actionPerformed(ActionEvent e) {
  // 2.1.1 - invoca o método da fachada.
    System.out.println(“O botão salvar foi pressionado!”);
 }
}
```

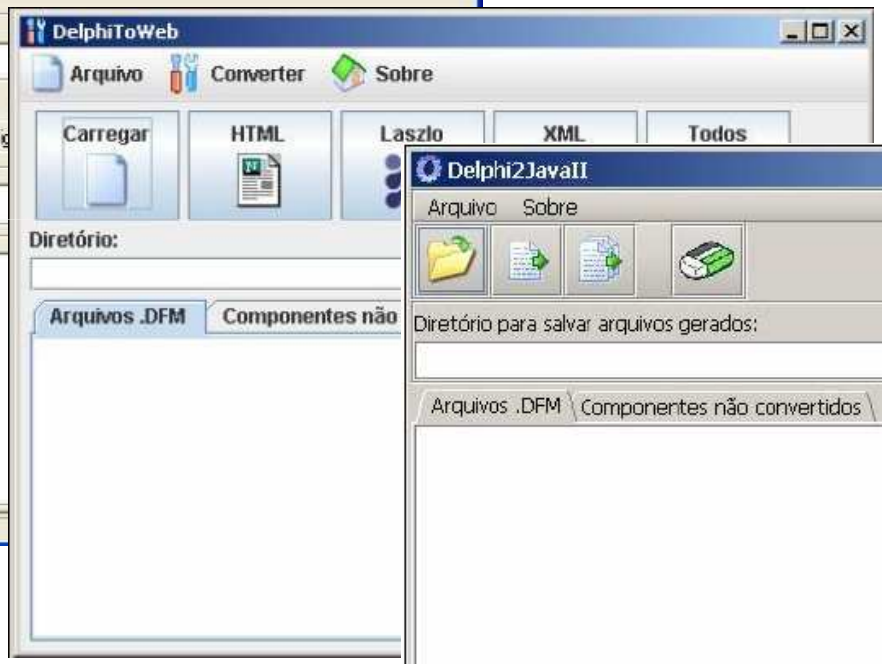
Delphi2Java-II

- O que é?
- Histórico
  - Fonseca 2005
  - Souza 2005
  - Silveira 2006
- Funcionalidades da ferramenta

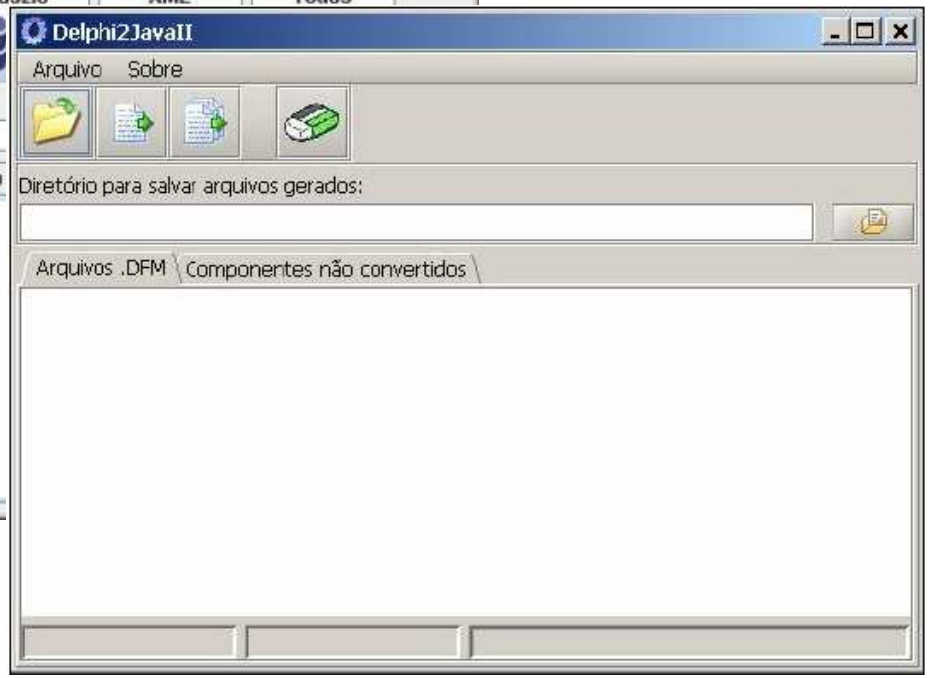
Delphi2Java-II



Fonseca 2005



Souza 2005



Silveira 2006

Trabalhos Correlatos

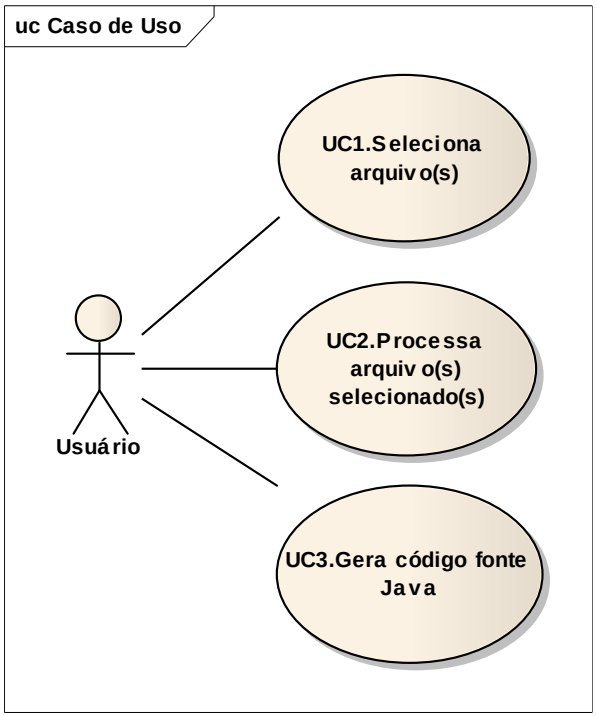
- Ferramenta para cálculo de métricas em software orientado a objetos
- Delphi2Java
- Ferramenta conversora de interfaces gráficas: Delphi2Java-II
- Ferramenta para conversão de formulários Delphi em páginas HTML
- Extensão da ferramenta Delphi2Java-II para suportar componentes de banco de dados.
- Gerador de código Java a partir de arquivos do Oracle Forms 6I
- Delphi2CS

Desenvolvimento

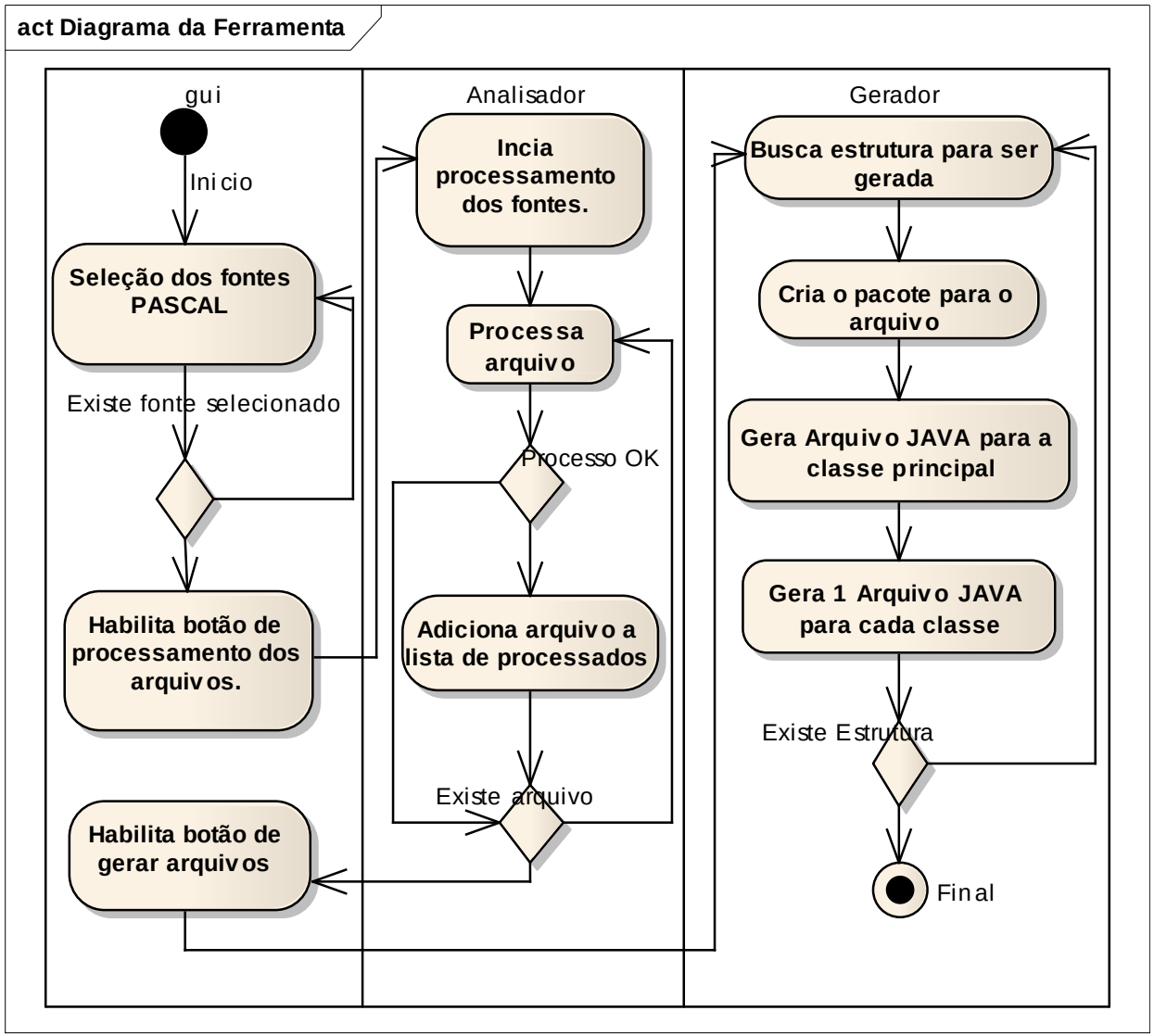
- Especificação
  - Etapas do processo
- Metalinguagem
- Diagrama de Classes
- Implementação
  - Técnicas e Ferramentas
  - Processo Inicial
  - Análise do fonte Pascal
  - Geração do código Java

Especificação

- Seleção dos Arquivos
- Processamentos dos Fontes
- Montar Código fonte Java



Etapas do processamento



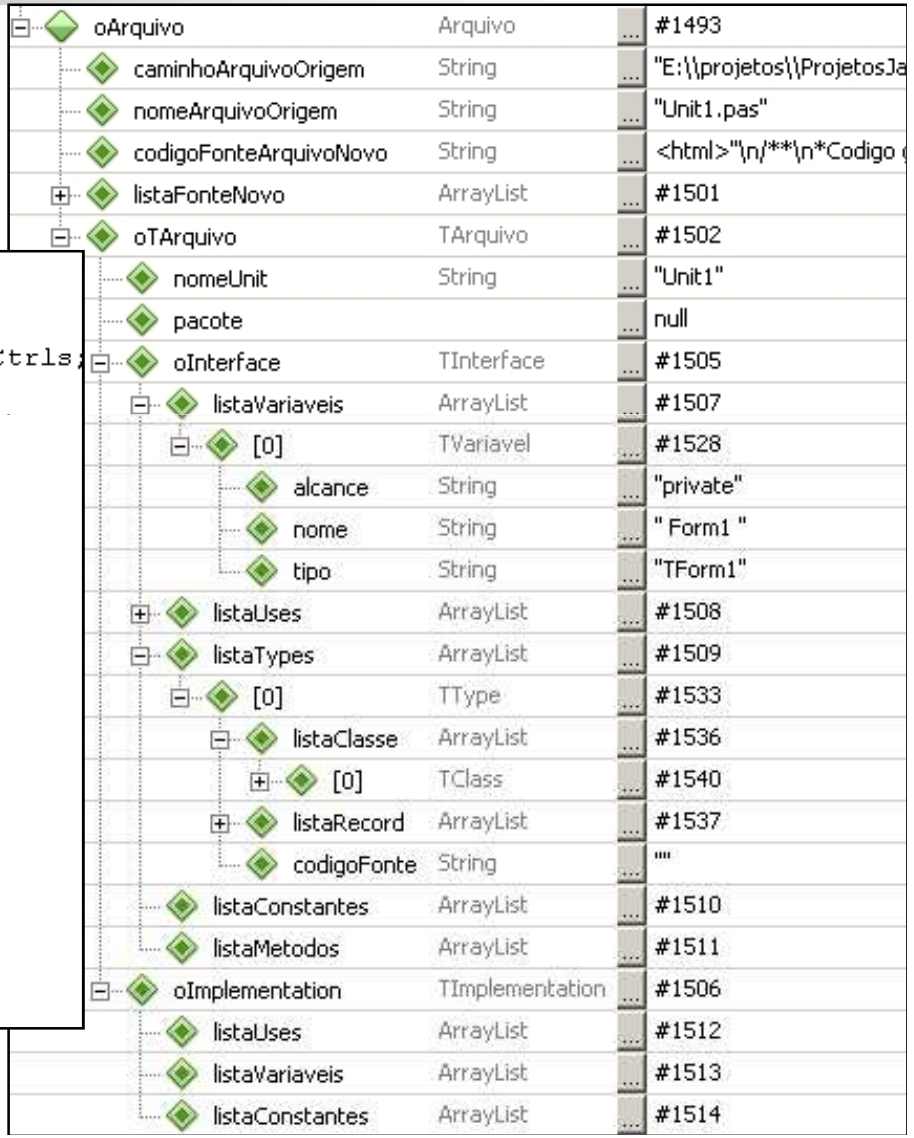
Metalinguagem

- O que é uma Metalinguagem?
  - Estrutura para armazenar informações.
- Metalinguagem no contexto do trabalho
  - Armazena informações sobre o fonte analisado.
- Estrutura da metalinguagem

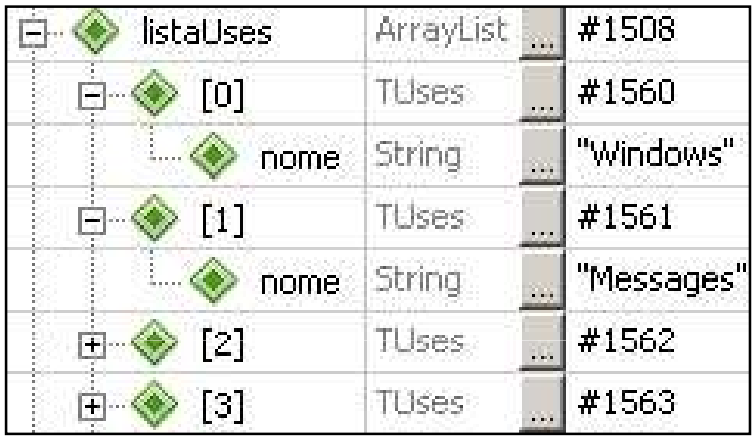
TArquivo	TInterface	TImplementation
TUses	TConstante	TProperty
TType	TClass	TRecord
TVariavel	TMetodos	TBloco
TComandos	TFor	TWhile
TRepeat	TCase	TOpcaoCase
TIf	TProcedimento	TAtribuicao
TComparacao		

Metalinguagem

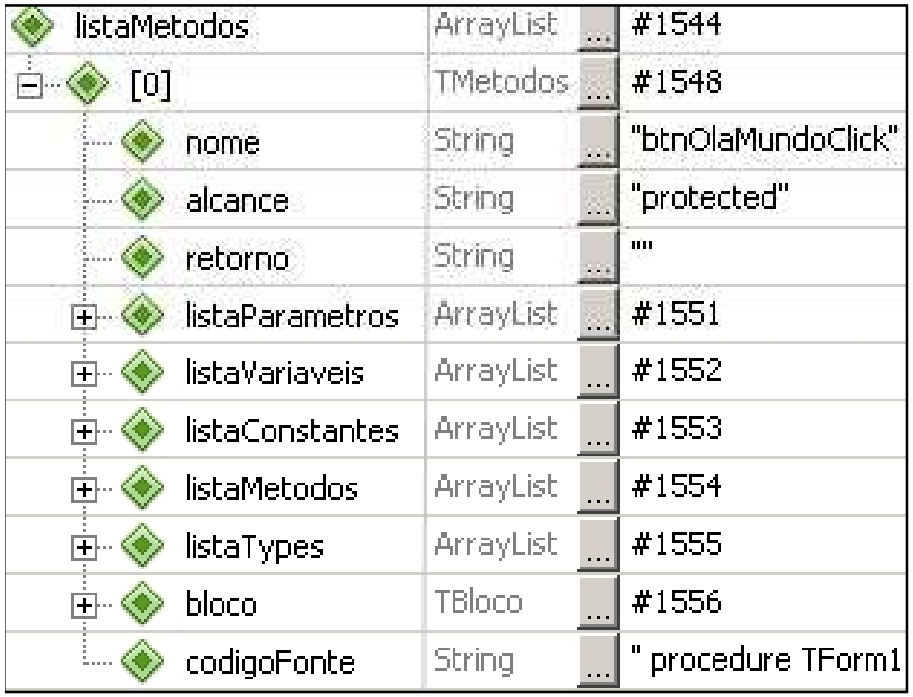
```
unit Unit1;
interface
uses
Windows,Messages,Classes,Graphics,Controls,Forms,Dialogs,StdCtrls;
type
  TForm1 = class(TForm)
    btnOlaMundo: TButton;
    procedure btnOlaMundoClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.btnOlaMundoClick(Sender: TObject);
var ind: integer;
begin
  ind:= 0;
  ShowMessage('Olá Mundo!');
  ind:= ind + 1;
end;
end.
```



Metalinguagem



Metalinguagem



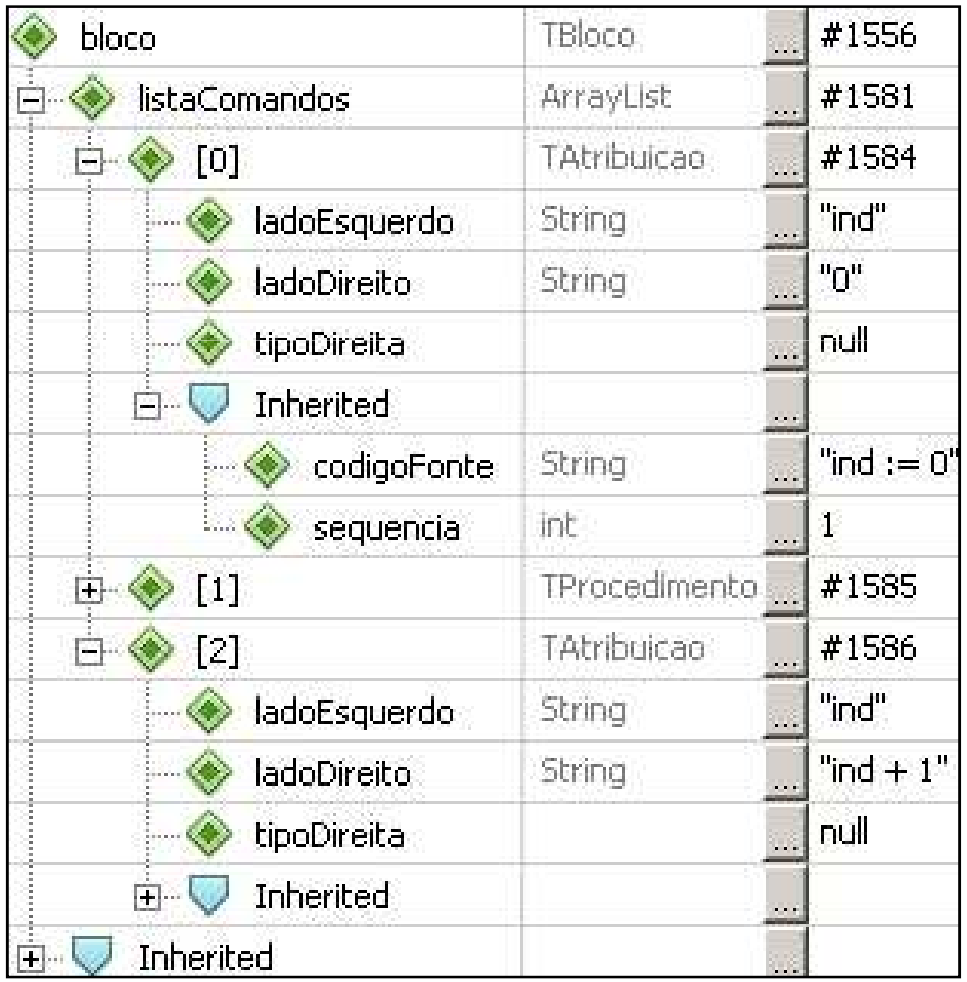


Diagrama de Classes

- Pacotes da aplicação

Pacote Gui

Pacote ParserArquivo

Pacote Compilador

Pacote Classes

Pacote Classes

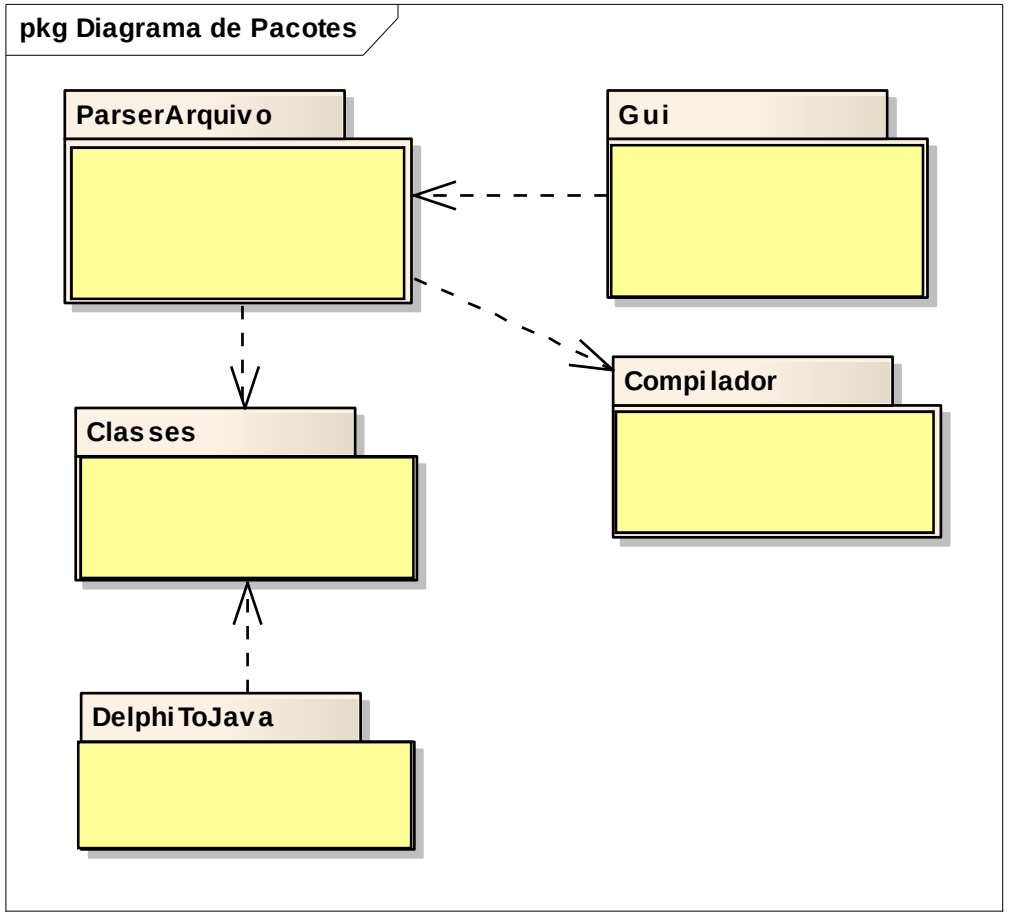


Diagrama de Classes

- Pacote parserarquivo

Classes
ConverteArquivo
Analisador
Arquivo
Gerador
NovoLexico
FonteNovo
Util

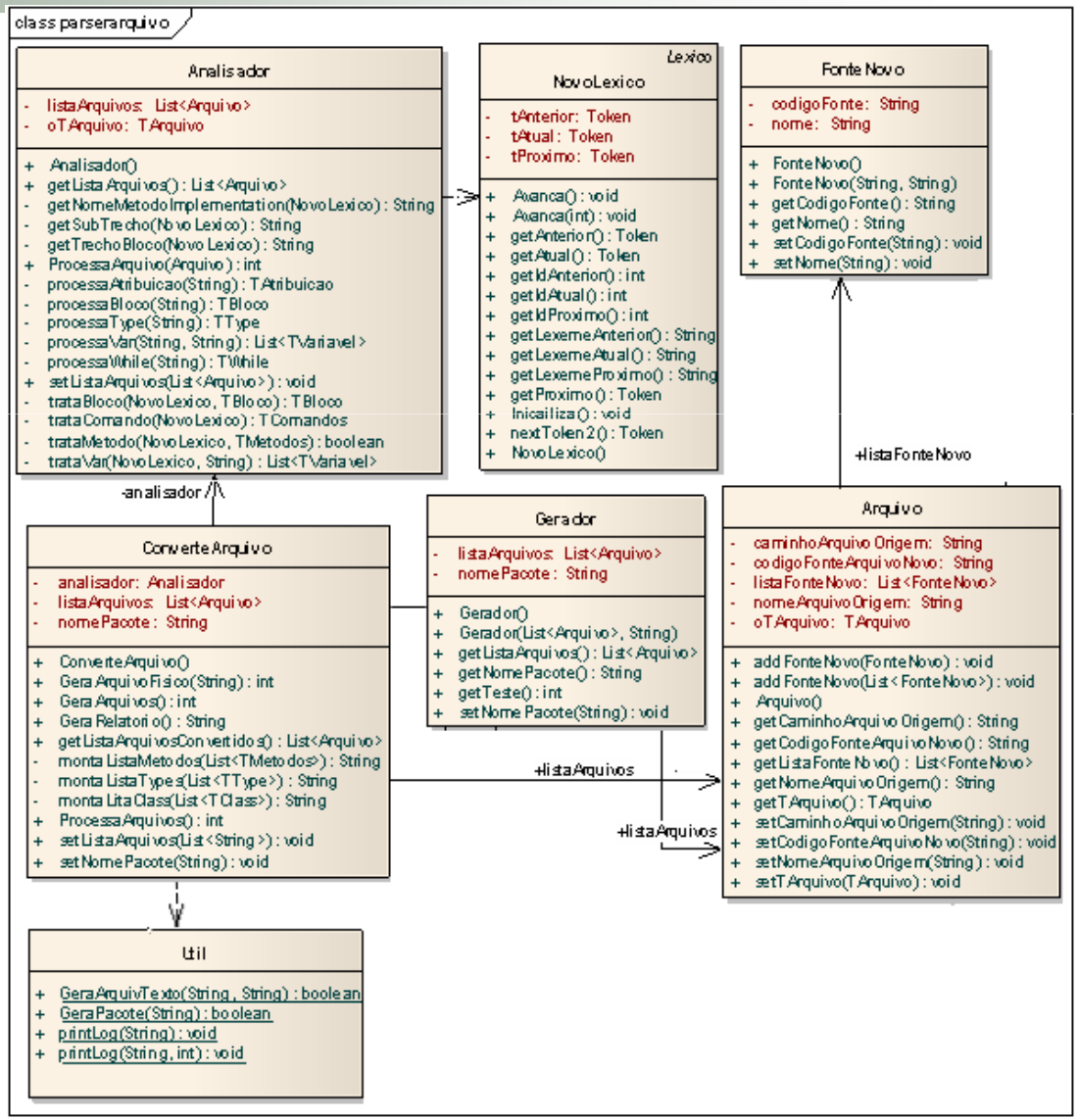
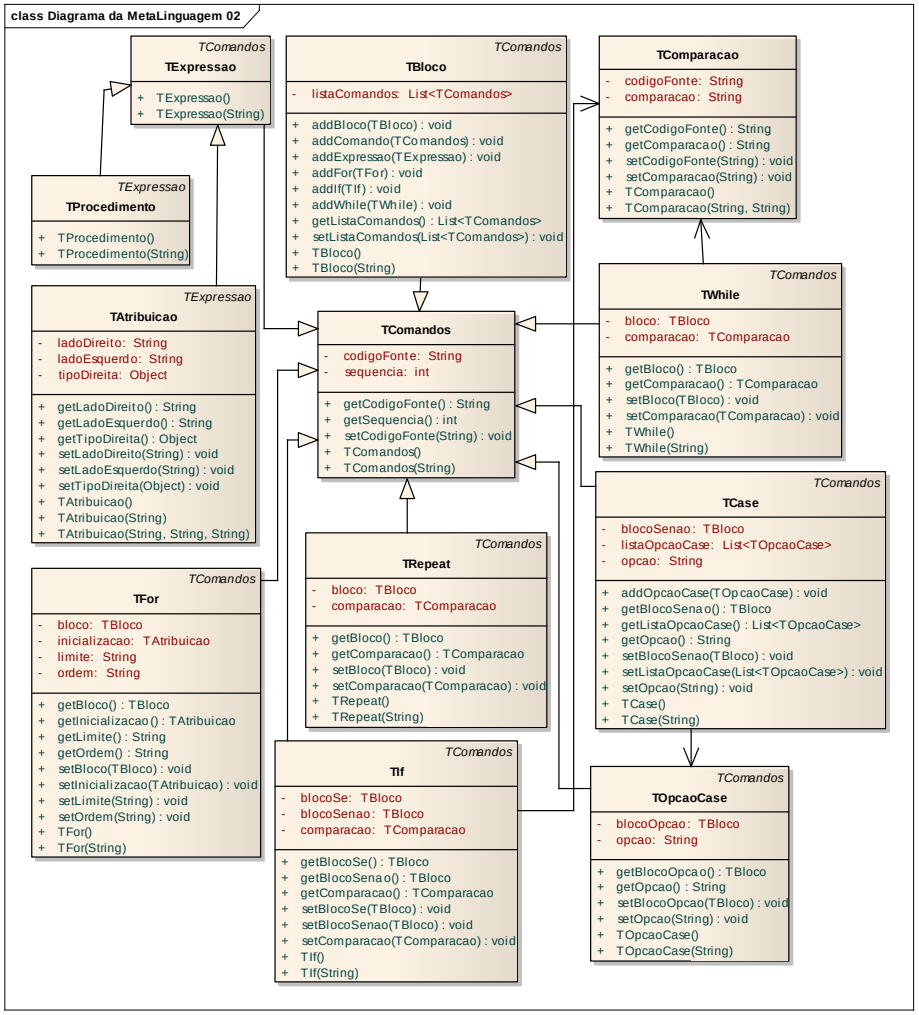


Diagrama de Classes

- Pacote classes



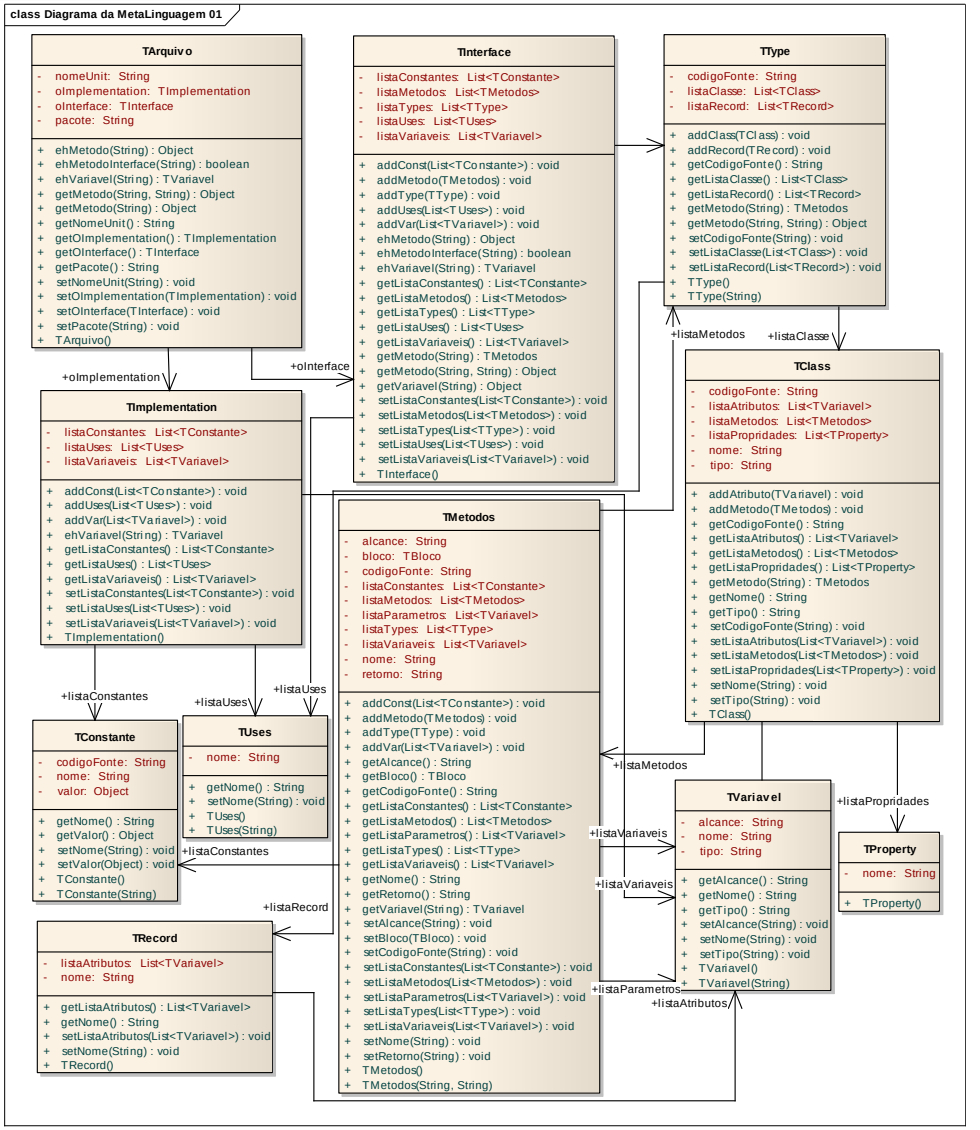
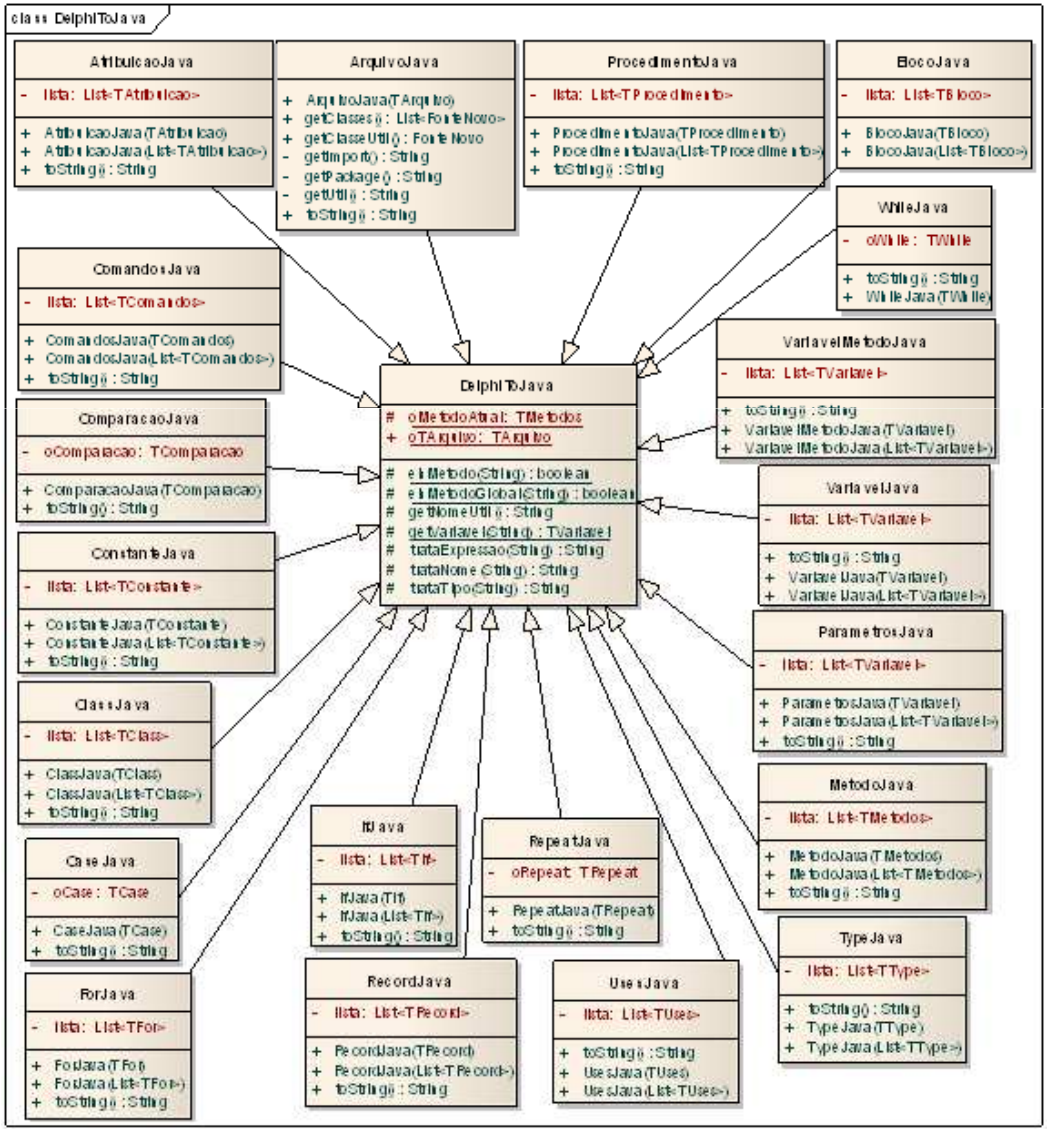


Diagrama de Classes

- Pacote DelphiToJava

Classes	
DelphiToJava	ArquivoJava
BlocoJava	AtribuicaoJava
Procedimentojava	ComandosJava
WhileJava	ForJava
ComparacaoJava	VariavelJava
VariavelMetodoJava	ConstanteJava
ParametrosJava	ClassJava
MetodoJava	CaseJava
IfJava	RepeatJava
RecordJava	UsesJava
TypeJava	

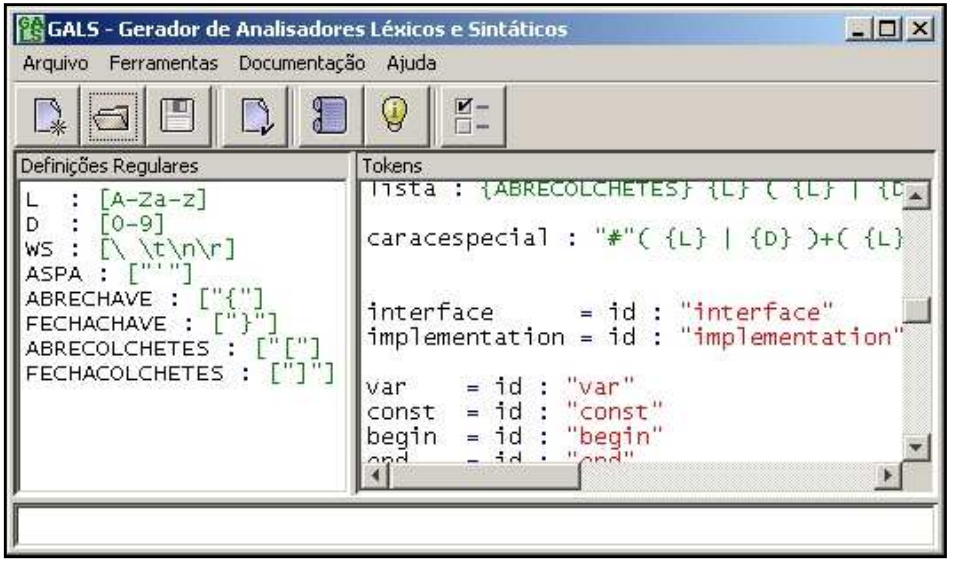


Implementação

- **Objetivo:** O propósito é ler um fonte PASCAL e converter os tratadores de eventos para a linguagem JAVA. De forma que essa conversão seja capaz de contemplar o maior número de rotinas que compõem os tratadores de evento do fonte PASCAL.

Técnicas e Ferramentas

- Enterprise Architect 7.0
- NetBeans 6.1
- JDK 1.6
- UML
- GALS



Processo Inicial

- Selecionar fontes para conversão
- Montagem da lista contendo os caminhos para os arquivos que serão analisados

Analise do fonte PASCAL

- Leitura da lista de fontes PASCAL
  - Classe ConverteArquivo

```
{...}
public void setListaArquivos(List<String> listaCaminhoArquivo){
try {
   this.listaArquivos.clear();//limpa a lista.
   //varre a lista de caminhos.
   for (String caminhoArquivo : listaCaminhoArquivo) {
      File f = new File(caminhoArquivo);
      if (f.exists()) {
         Arquivo arq = new Arquivo();
         //Atribui o caminho para ao arquivo.
         arq.setCaminhoArquivoOrigem(f.getAbsolutePath());
         arq.setNomeArquivoOrigem(f.getName());
         //Adiciona o objeto do tipo Arquivo a lista.
         this.listaArquivos.add(arq);
      }
   }
 {..}
 }
 {..}
```

Analise do fonte PASCAL

- Inicio da analise do fonte PASCAL
  - Classe ConverteArquivo

```
public int ProcessaArquivos() {
    if (this.listaArquivos.isEmpty()) {
        return 0;
    }
    this.analisador = new Analisador();
    for (Arquivo oArquivo : this.listaArquivos) {
        if (analisador.ProcessaArquivo(oArquivo) == 1) {
        } else {
            Util.printLog("Erro no Processamento do arquivo :", 1);
            Util.printLog(oArquivo.getCaminhoArquivoOrigem(), 1);
        }
    }
    return 1;
}
```

  - Classe Analisado

Analise do fonte PASCAL

- Para cada fonte selecionado para conversão é aplicada a seguinte regra:
  - Divisão do fonte em três partes:
    - `Unit`
    - *Interface*
    - *Implementation*
  - Varre cada umas das partes extraindo informações para carregar a metalinguagem

Analise do fonte PASCAL

- Extrai o nome da *Unit*
  - Atribui o nome da `Unit` a classe `TArquivo`
- Extrai da *interface* as declarações de:
  - `Uses`
  - `Types`
  - Variáveis
  - Constantes
  - Procedimentos
  - Funções

Analise do fonte PASCAL

- Analisador Léxico da *Interface*
  - Recebe o trecho de código da *interface*
  - Varre os tokens em busca de informações chave

```
private TInterface processaInterface(String fonte){
  try {
   NovoLexico lexico = new NovoLexico();
   lexico.setInput(fonte);
   lexico.Inicializa();
   TInterface oInterface = new TInterface();
   while (lexico.getAtual() != null) {
    switch (lexico.getIdAtual()) {
      case Constants.t_uses:
         String  uses = getTrechoUses(lexico);
         List<TUses> listaUses = null;
         listaUses = processaUses( uses);
         if (listaUses != null) {
             oInterface.addUses(listaUses);
         }
         break;
      case Constants.t_type:
         String _type = this.getTrechoType(lexico);
         TType oType = null;
         oType = this.processaType( type);
         if (oType != null) {
             oInterface.addType(oType);
         }
         break;
```

Analise do fonte PASCAL

- Buscando trecho de código de um subitem
  - Monta o trecho de código
  - Retorna o código para ser processado

```
private String getTrechoType(NovoLexico lexico) {
 String retorno = "";
 boolean ehFora = false;
 try {
  if (lexico.getAtual() != null) {
    if (lexico.getIdAtual() == Constants.t_type) {
        lexico.Avanca();
    }
    do {
     retorno += " " + lexico.getLexemeAtual();
     lexico.Avanca();
     if (lexico.getIdAtual() == Constants.t_class ||
             lexico.getIdAtual() == Constants.t_record) {
         ehFora = false;
     } else if (lexico.getIdAtual() == Constants.t_end) {
         ehFora = true;
     }
    } while(lexico.getIdProximo()!= Constants.t_type &&
           (lexico.getIdProximo()!= Constants.t_const || !ehFora)&&
           (lexico.getIdProximo()!= Constants.t_var || !ehFora) &&
           (lexico.getIdProximo()!= Constants.t_procedure||!ehFora)&&
           (lexico.getIdProximo()!= Constants.t_function||!ehFora)&&
           (lexico.getIdProximo()!= Constants.t_implementation));
    return retorno;
```

Analise do fonte PASCAL

- Analise léxica do *Type*
  - Recebe o trecho de código
  - Varre os tokens em busca de informações chave

```
private TType processaType(String fonte) {
 try {
  String nome = "";
  TType oType = new TType();
  NovoLexico lexico = new NovoLexico();
  lexico.setInput(fonte);
  lexico.Inicializa();
  while (lexico.getAtual() != null) {
   switch (lexico.getIdAtual()) {
    case Constants.t_id:
      nome = lexico.getLexemeAtual();
      break;
    case Constants.t_class:
      if (lexico.getIdAnterior() == Constants.t_igual) {
        String _class = "";
        _class = nome;
        _class += " " + lexico.getLexemeAnterior();
        do {
            _class += " " + lexico.getLexemeAtual();
            lexico.Avanca();
        } while (lexico.getIdAnterior() != Constants.t_end);
        if (!_class.trim().equals("")) {
            TClass oClass = this.processaClass( class);
            if (oClass != null)
        }       oType.addClass(oClass);
        nome = "";
      }
      break;
    case Constants.t_record:
        if (lexico.getIdAnterior() == Constants.t_igual) {
            String _record = "";
            _record = nome;
```

Analise do fonte PASCAL

- Analise Léxica da *Implementation*
  - Recebe o trecho de código da *implementation*
  - Varre os tokens em busca de informações chave

```
private TImplementation processaImplementation(String fonte) {
  String alcance = "protected";
  TImplementation oImplementation = new TImplementation();
  try {
    NovoLexico lexico = new NovoLexico();
    lexico.setInput(fonte);
    lexico.Inicializa();
    while (lexico.getAtual() != null) {
      switch (lexico.getIdAtual()) {
        case Constants.t_uses:
          String _uses = getTrechoUses(lexico);
          ...
        case Constants.t_var:
          List<TVariavel> listVar = this.trataVar(lexico, alcance);
          if (listVar != null)
              oImplementation.addVar(listVar);
          break;
        case Constants.t_const:
          String _const = getTrechoConst(lexico);
          ...
        case Constants.t_procedure:
          String metodoP = getTrechoMetodo(lexico);
          processaMetodoImplementation(metodoP);
          break;
        case Constants.t_function:
          String metodoF = getTrechoMetodo(lexico);
          processaMetodoImplementation(metodoF);
          ...
  }
  return oImplementation;
```

Analise do fonte PASCAL

- Analise léxica dos métodos na *Implementation*
  - Busca o trecho de código da *implementation*
  - Varre os tokens em busca de informações chave

```
private String getTrechoMetodo(NovoLexico lexico) {
 String retorno = "";
 int qtdBegin = 0;
 int qtdEnd = 0;
 int qtdMet = 0;
 try {
  do {
    if (lexico.getIdAtual() == Constants.t_procedure||
        lexico.getIdAtual() == Constants.t_function) {
        qtdMet++;
    }
    if (qtdBegin == qtdEnd && qtdBegin > 0) {
        qtdMet--;qtdBegin--;qtdEnd--;
        if (qtdMet == 0)
            break;
    }
    if (lexico.getIdAtual() == Constants.t_begin ||
        lexico.getIdAtual() == Constants.t_case ||
        lexico.getIdAtual() == Constants.t_except ||
        lexico.getIdAtual() == Constants.t_finally) {
        qtdBegin++;
    }
    if (lexico.getIdAtual() == Constants.t_end)
        qtdEnd++;
    retorno += " " + lexico.getLexemeAtual().toString();
    lexico.Avanca();
  } while (lexico.getAtual() != null);
...
```

```
private void processaMetodoImplementation(String fonte){
 String nomeClass = "";
 try {
  String nomeMetodo = "";
  if (fonte.trim().length() <= 0)
      return;
  TMetodos oMetodo = null;
  NovoLexico lexico = new NovoLexico();
  lexico.setInput(fonte);
  lexico.Inicializa();
  if (lexico.getAtual() == null)
      return;
  String temp = this.getNomeMetodoImplementation(lexico);
  if (temp.trim().equals(""))
      return;
  nomeClass = temp.trim().split("\\.")[0];
  nomeMetodo = temp.trim().split("\\.")[1];
  oMetodo = oTArquivo.getMetodo(nomeClass, nomeMetodo);
  if (oMetodo != null) {
      oMetodo.setCodigoFonte(fonte);
  } else
      return;
  this.trataMetodo(lexico, oMetodo);
...
```

Geração de código Java

- Objetivo
- Método utilizado para geração
  - Usar como base a metalinguagem
  - Classes que tem conhecimento
- Estrutura dos arquivos gerados
  - Pasta com o nome da `unit` do arquivo
  - Nome da `unit` + “_Util.java”
  - Nome da classe ou record + “.java”

Geração de código Java

- Inicio do processo
  - Leitura das estruturas geradas
- Pacote DelphiToJava
  - Classe ArquivoJava
- Varre a metalinguagem

```
public int ConverteArquivos() {
  for(Arquivo oArquivo :this.getListaArquivos()) {
    ArquivoJava oArquivoJava = new ArquivoJava(oArquivo.getTArquivo());
    String fonte = oArquivoJava.toString();
    oArquivo.addFonteNovo(oArquivoJava.getClasses());
    oArquivo.setCodigoFonteArquivoNovo(fonte);
  }
  return 1;
}
```

```
public String toString(){
 String retorno = this.getCabecalho();
 retorno += this.getPackage();
 retorno += this.getImport();
 retorno += this.getUtil();
 retorno += (new TypeJava(oTArquivo.getOInterface().getListaTypes())).toString();
 return retorno;
}
```

Geração de código Java

- Um detalhamento do processo
  - `TypeJava ClassJava`

```
public String toString() {
 String retorno = "";
 retorno += "\n //Type";
 for (TType oType : this.lista) {
  retorno += "\n";
  retorno +=(new RecordJava(oType.getListaRecord())).toString();
  retorno +="\n";
 } retorno +=(new ClassJava(oType.getListaClasse())).toString();
 retorno += "\n //FIM Type";
 return retorno;
}
```

```
public String toString() {
 String retorno = "";
 for (TClass oClass : this.lista) {
  retorno +="\n  //Inicio Classe " + oClass.getNome() + ".";
  retorno +="\n  public class " +oClass.getNome()+" extends "+oClass.getTipo()+" {";
  retorno +=(new VariavelJava(oClass.getListaAtributos())).toString();
  retorno +=(new MetodoJava(oClass.getListaMetodos())).toString();
  retorno +="\n  }//Final Classe " + oClass.getNome() + ".";
 }
 return retorno;
```

Resultados

- Os resultados vem a acrescentar ao projeto Delphi2Java-II uma solução para conversão dos tratadores de eventos
- Todas as estruturas mapeadas na metalinguagem foram transcritas para o fonte JAVA

Resultados

- Fonte PASCAL

```
unit Unit1;
interface
uses
Windows,Messages,Classes,Graphics,Controls,Forms,Dialogs,StdCtrls;
type
  TForm1 = class(TForm)
    btnOlaMundo: TButton;
    procedure btnOlaMundoClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
  pessoa = record
    nome : String;
    idade : integer;
  end;
procedure Mensagem;
var
  Form1: TForm1;
implementation
procedure TForm1.btnOlaMundoClick(Sender: TObject);
var ind: integer;
    nova : pessoa;
begin
  ind:=1;
  Mensagem;
  nova.nome := 'João';
  nova.idade := 52;
end;
procedure Mensagem;
begin
    ShowMessage('Olá Mundo!');
end;
end.
```

Resultados

- Fontes gerados em Java

```
package Unit1;
 // *NP* import Windows;
 // *NP* import Messages;
...
//Classe principal,contem os métodos globais.
 public class Unit1_Util {
   private TForm1 form1;
   public static void mensagem() {
     showMessage("Olá Mundo!");
   }//metodo(Mensagem)
}//class Util1.
```

```
package Unit1;
//Record pessoa.
  public class pessoa{
    protected String nome;
    protected int idade;
}//Record
```

```
package Unit1;
//Início Classe TForm1.
public class TForm1 extends TForm{
  protected TButton btnOlaMundo;
  protected void btnOlaMundoClick(TObject sender){
    int ind;
    pessoa nova;
    ind = 1;
    //chamada de método da classe Util.
    Unit1_Util.mensagem();
    inc(ind);
    nova.nome = "João";
    nova.idade = 52;
 }//metodo(btnOlaMundoClick)
}//Final Classe TForm1.
```

Conclusões

- Utilizada parcialmente a BNF da linguagem Object Pascal
  - Uso parcial da BNF da linguagem Object Pascal em função da sua complexidade
- Resultados satisfatórios na utilização somente do analisador léxico
- Ganhos com o uso da Metalinguagem
  - Ganhos com o uso da metalinguagem permitindo não só a geração de código como a assimilação do mesmo (Mattos,2003).

Extensões

- Conversão sintática e semântica das bibliotecas de run-time do Delphi para bibliotecas de run-time do Java
- Conversão sintática e semântica de outros componentes do Delphi
- Integração das rotinas do protótipo com o projeto Delphi2Java-II