MOTOR DE *TEMPLATES* PARA DELPHI

Acadêmico: Thiago Kewitz Demarchi

Orientadora: Joyce Martins

Roteiro

- Introdução
  - Objetivos do trabalho
- Fundamentação teórica
  - Conceitos básicos
  - Trabalhos correlatos
- Desenvolvimento da biblioteca
  - Requisitos do motor de *templates*
  - Especificação (da linguagem de *templates* e do motor de *templates*)
  - Implementação
  - Estudos de caso
  - Resultados e discussão
- Conclusão
  - Extensões

Introdução

- Demanda por maior flexibilidade nos softwares
- Flexibilidade pode ser obtida através de *templates*
- Mais vantagens:
  - Desenvolvimento simplificado
  - Redução de custos
  - Desenvolvimento paralelo da formatação de saída e da lógica
- Motores de *templates*:
  - Específicos
    - Módulos da aplicação que atendem restritamente as suas necessidades
  - Genéricos
    - Bibliotecas que disponibilizam *templates* para aplicações

Objetivos do trabalho

- Desenvolver um motor de *templates* genérico para softwares desenvolvidos em Delphi 7

- Disponibilizar uma biblioteca com classes, atributos e métodos que implementem o motor de *templates*

- Permitir mapear os dados da aplicação para serem referenciados nos *templates*

- Analisar os *templates* léxica e sintaticamente, reconhecendo o código estático e o código dinâmico, bem como seus elementos

- Interpretar e substituir o código dinâmico do *template* conforme os dados mapeados

Fundamentação teórica

- Conceitos básicos
  - *Templates* e suas aplicações
  - Motores de *templates*
  - Processadores de linguagem
- Trabalhos correlatos
  - Smarty
  - FastTrac
  - Velocity

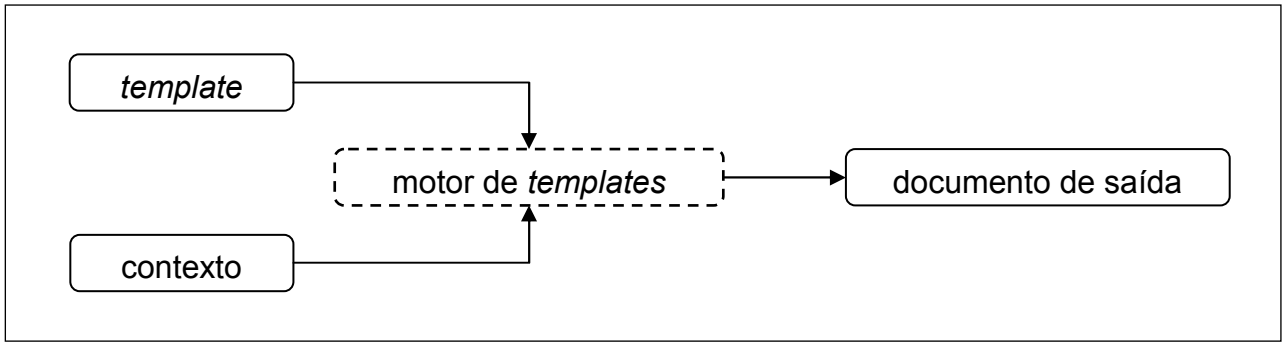
Templates e suas aplicações

- Arquivos base para geração de textos formatados
- Permitem separar a lógica e a formatação de saída
- Formados por:
  - Código estático
  - Código dinâmico

- Utilizados em geradores de código
  - Programas que escrevem programas
  - Mudanças globais em alterações únicas

Motores de *templates*

- Geram documentos de saída com base em *templates*
- Dados da aplicação mapeados em um contexto



- Interpretação através de linguagem de *templates*
  - Provê:
    - Substituição de valores
    - Estruturas de controle complexas (laços e comandos condicionais)

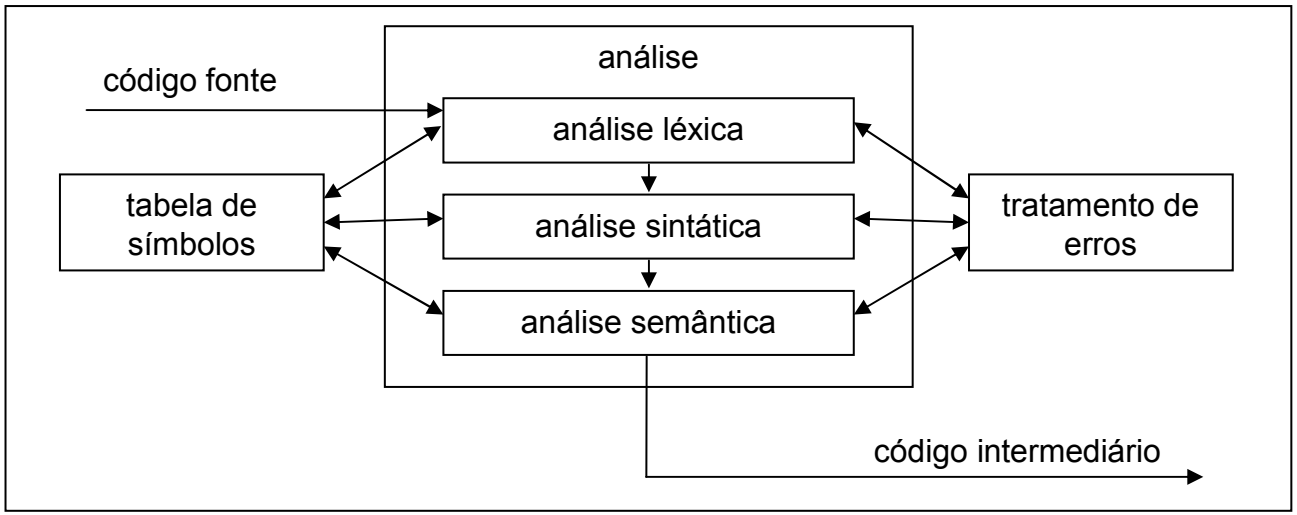
Processadores de linguagem

- Usados no reconhecimento e interpretação da linguagem de *templates*
- Responsáveis por:
  - Ler entradas de texto fonte
  - Reconhecer o significado de suas construções (módulo de análise)
  - Executar ações em função destes significados (módulo de síntese)

- Análise:
  - Léxica
  - Sintática
  - Semântica

Processadores de linguagem (continuação)

- Arquitetura básica do módulo de análise



Trabalhos correlatos

- Smarty
  - PHP
  - Focado na web, separar design e lógica
- FastTrac
  - Encapsulamento do Smarty para Delphi
  - Requer um conjunto de bibliotecas na aplicação
- Velocity
  - Java
  - VTL – linguagem de *templates* simples
    - Manipulação de variáveis
    - Diretivas de controle de fluxo
    - Reuso de código

Desenvolvimento da biblioteca

- Requisitos do motor de *templates*
- Especificação da linguagem de *templates*
- Especificação do motor de *templates*
  - Diagrama de casos de uso
  - Diagrama de classes
- Implementação
- Estudos de caso
  - Gerador de código de fonte
  - Catálogo de telefones
- Resultados e discussão

Requisitos do motor de *templates*

↳ Requisitos funcionais e não funcionais

REQUISITOS FUNCIONAIS
RF01: Permitir ao programador mapear dados de sua aplicação a um contexto.
RF02: Analisar léxica e sintaticamente arquivos de *template*, reconhecendo código estático e código dinâmico.
RF03: Gerar saída com base no *template*, interpretando e substituindo o código dinâmico conforme os dados mapeados no contexto.
RF04: Disponibilizar uma linguagem de *templates* com elementos que permitam: declaração e uso de variáveis, controle de fluxo de execução e reuso de código.

REQUISITOS NÃO FUNCIONAIS
RNF01: Ser implementado no ambiente Delphi 7.
RNF02: Ser compatível e permitir sua inclusão e uso em aplicações desenvolvidas em Delphi 7.
RNF03: Possuir funcionalidades em sua linguagem de *templates* semelhantes às da VTL do Velocity.

Especificação da linguagem de *templates*

- BNF
- Gramáticas estudadas:
  - Motor de *templates* específico de Orsi (2006)
    - Não utilizado
  - VTL – Velocity
    - Utilizado como base para a linguagem de *templates*
- Recursos:
  - Declaração e uso de variáveis
  - Controle de fluxo de execução
  - Reuso de código

Especificação da linguagem de *templates* (continuação)

Recursos da linguagem de *templates*

RECURSO	EXEMPLO
declaração e atribuição de variáveis	`#set ($valor = "true")`
referência	`$nome  $pessoa.nome  $pessoa.nome()`
bloco de comandos	`#begin ... #end`
controle de fluxo – condicional	`#if ($valor == 1) ... #elseif ($valor == 2) ... #else ... #end`
controle de fluxo – repetição	`#foreach ($item in $lista) ... #end`
controle de fluxo – parada	`#stop`
inclusão de arquivos – texto	`#include("texto.txt")`
inclusão de arquivos – *template*	`#parse("templAux.vm")`
reuso de código – definição de macro	`#macro(mcTeste $arg1 $arg2) ... #end`
reuso de código – invocação de macro	`#mcTeste(1 "teste")`
comentário – linha	`## comentário em linha`
comentário – bloco	`#* comentário em várias linhas *#`

Especificação da linguagem de *templates* (continuação)

- Exemplo de *template*

```
#foreach ($objeto in $lista)

  #set ($forma = "")

  #if ($objeto.ehQuadrado)
    #set ($forma = "quadrado")
  #elseif ($objeto.ehRedondo)
    #set ($forma = "redondo")
  #else
    #set ($forma = "outra")
  #end

  Objeto: $objeto.nome
  Forma: $forma

#end
```

Especificação do motor de *templates*

- Enterprise Architect
- Orientação a objetos
- Diagramas UML:
  - Casos de uso
  - Classes

Diagrama de casos de uso

- Casos de uso

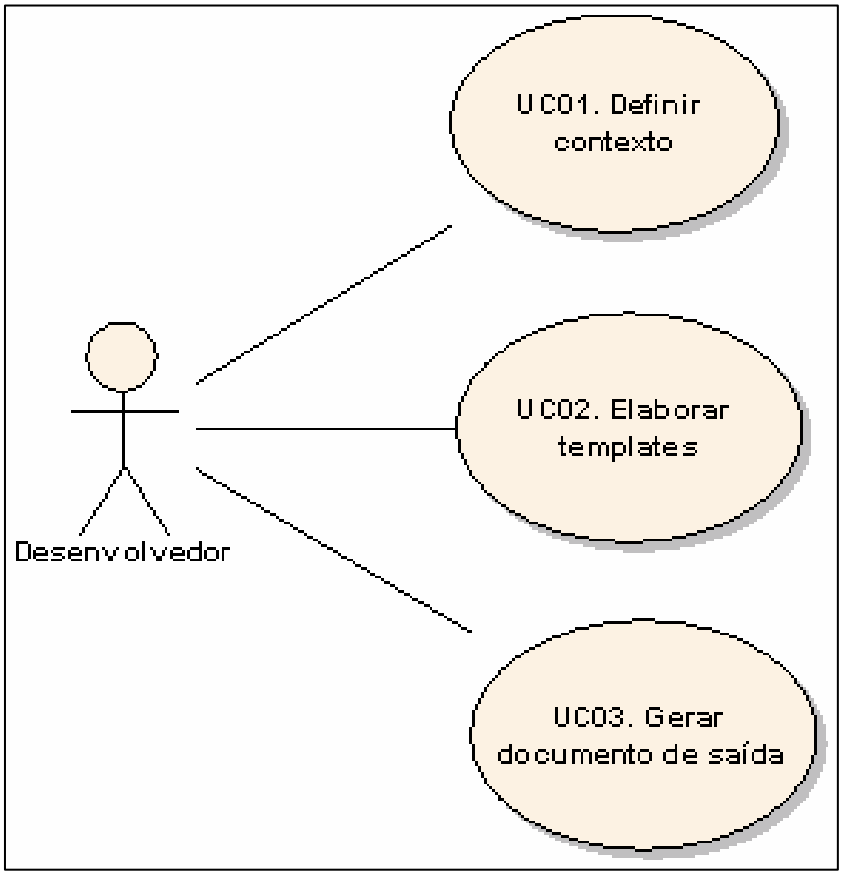
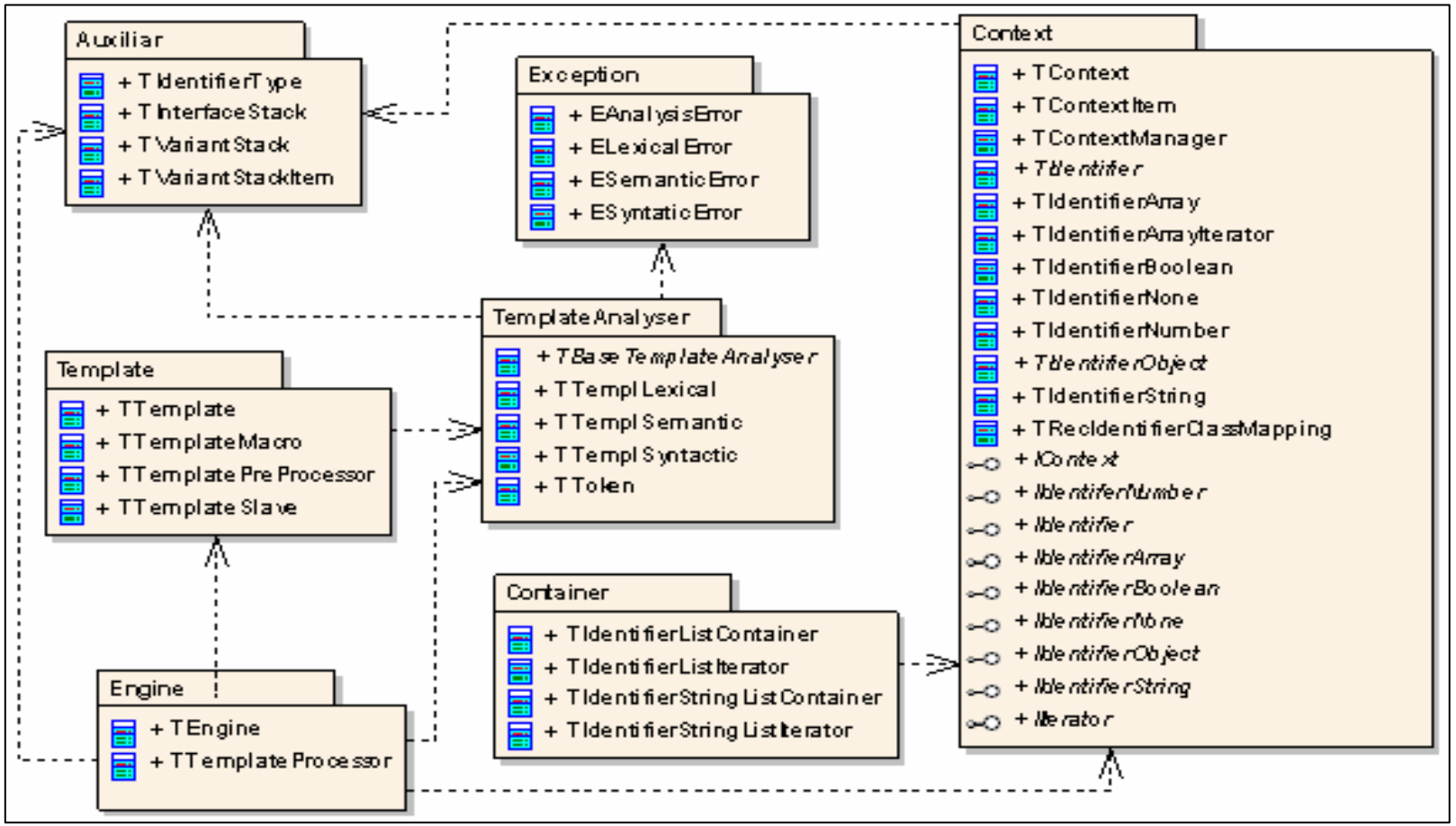


Diagrama de classes

Pacotes de classes



Implementação

- Object Pascal
- Delphi 7
- GALS

- Preocupação com a qualidade
  - Ferramenta para teste – TestCASE
  - Cenários de teste contendo:
    - *Template*
    - Resultado esperado (sucesso ou erro)
  - Possibilidade de executar todos os cenários
  - Atualização e criação de cenários

Implementação (continuação)

- Geração dos analisadores léxico, sintático e semântico
- Processamento através de ações semânticas
- Dois processamentos (duas análises)
  - Pré-processamento
    - Levantamento de informações sobre macros
  - Processamento principal
    - Interpretação do *template* e geração da saída
- Contexto
  - Mapeamento de identificadores através de chaves únicas

Implementação (continuação)

- Identificadores:
  - Sem tipo ou nulo (necessidade interna)
  - Numéricos, alfanuméricos e booleanos
  - Vetoriais
    - Conjunto de outros identificadores
    - Iteração de valores
  - De objeto
    - Implementação básica é abstrata
    - Deriva em contêineres de objetos negócio
    - Disponibiliza propriedades e métodos
      - Implementados manualmente
      - Propriedades `published` disponíveis automaticamente
    - Mapeamento de classes contêineres
    - Classes contêineres para `TList` e `TStringList`

Implementação (continuação)

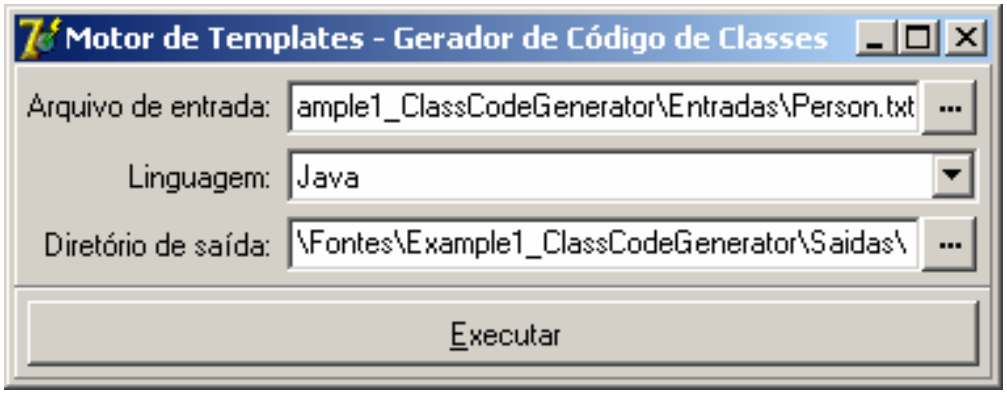
- Interface do motor de *templates*
  - Entradas:
    - *Template*
    - Contexto
  - Processamento
  - Saída de texto formatado
- Recursos tratados individualmente em ações semânticas
- Processamento auxiliar (interno)
  - Tratamento de mensagens e da parada de execução
- Código estático
  - Reconhecimento e transcrição
  - Formatação
    - Ignorada pelo analisador léxico
    - Regras de aceitação

Estudos de caso

- Dois estudos de caso:
  - Gerador de código fonte
  - Catálogo de telefones

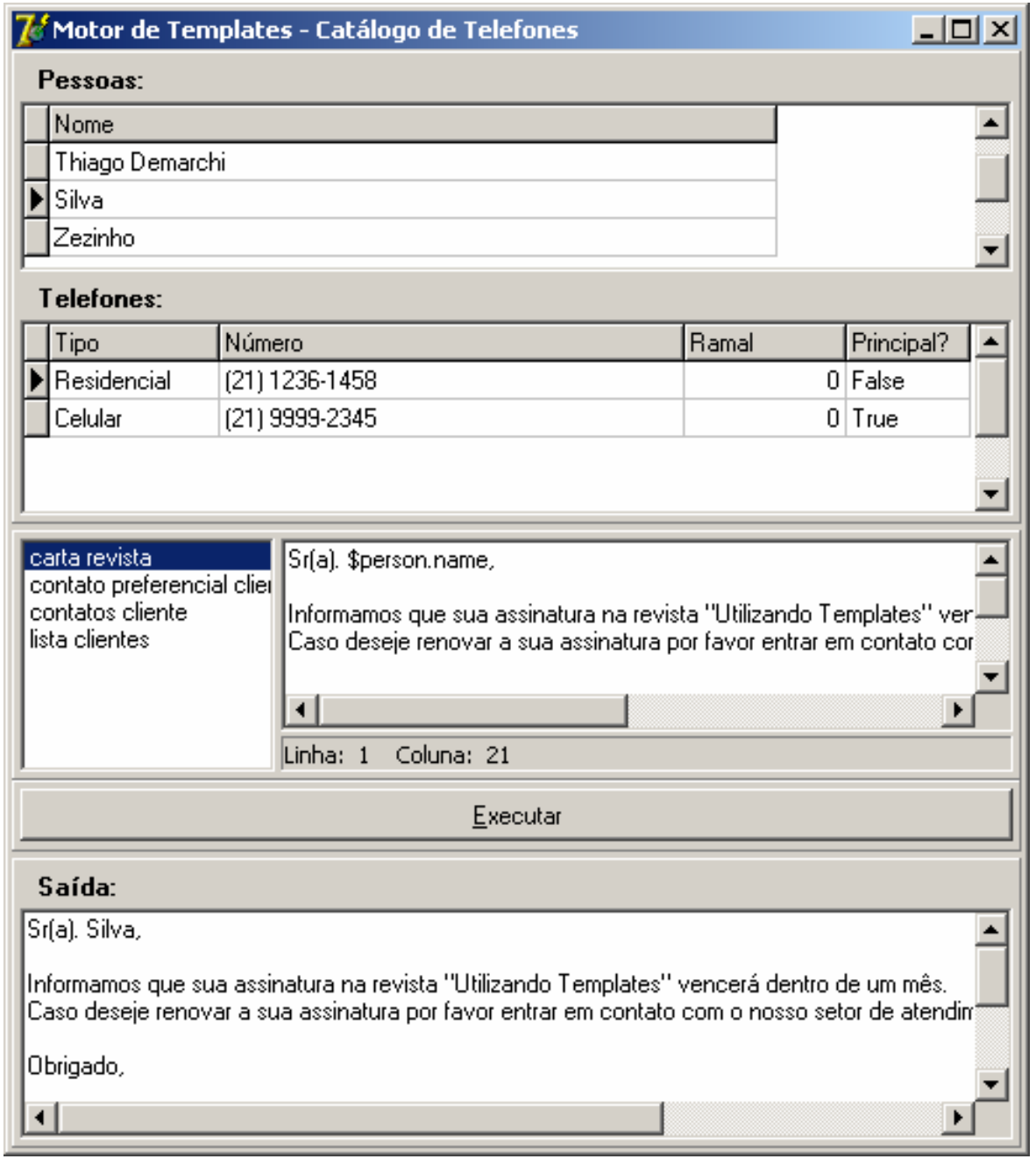
Estudos de caso (continuação)

- Gerador de código fonte
  - Leitura de arquivo texto de entrada
    - Dados de classes:
      - Nome da classe
      - Atributos alfanuméricos
  - Gera fonte das classes em uma linguagem



Estudos de caso (continuação)

- Catálogo de telefones
  - Leitura de arquivo texto de entrada
    - Dados de pessoas e telefones
      - Nome da pessoa
      - Telefones e detalhes
  - Gera documentos de saída
    - Com base nos dados do catálogo
    - Com base no *template* selecionado


Motor de Templates - Catálogo de Telefones
Pessoas:
Nome
Thiago Demarchi
Silva
Zezinho
Telefones:
Tipo
Número
Ramal
Principal?
Residencial
(21) 1236-1458
0
False
Celular
(21) 9999-2345
0
True
carta revista
contato preferencial clie
contatos cliente
lista clientes
Sr(a). $person.name,
Informamos que sua assinatura na revista "Utilizando Templates" ver
Caso deseje renovar a sua assinatura por favor entrar em contato cor
Linha: 1 Coluna: 21
Executar
Saída:
Sr(a). Silva,
Informamos que sua assinatura na revista "Utilizando Templates" vencerá dentro de um mês.
Caso deseje renovar a sua assinatura por favor entrar em contato com o nosso setor de atendim
Obrigado,

Resultados e discussão

- Atende aos requisitos propostos
- Motor de *templates* para aplicações Delphi 7
- Linguagem de *templates* similar a VTL do Velocity
  - Simples
  - Recursos: declaração de variáveis, controle de fluxo e reuso de código

- Comparativo

CARACTERÍSTICA	Smarty	FastTrac	Velocity	Motor de *templates* para Delphi
linguagem destino para uso da biblioteca	PHP	Delphi 6	Java	Delphi 7
utilização de bibliotecas terceiras para o funcionamento do motor de *templates*	não	sim	não	não
linguagem de *templates* simples	não	não	sim	sim

Conclusão

- Vantagens da utilização de um motor de *templates* genérico

- Interpretação de arquivos
  - Análise léxica, sintática e semântica
    - Código fonte gerado pelo GALS
    - Reconhecimento de *tokens*
    - Reconhecimento de código estático e dinâmico
    - Processamento de ações semânticas

- Informações da aplicação para o *template*
  - Mapeamento no contexto
  - Apelidos únicos
  - Objetos contêineres
    - Disponibilizar propriedades e métodos

Extensões

↳ *To-do list* com possíveis melhorias e sua complexidade:

Criação de um editor gráfico de *templates*.	10
Possibilidade de invocação de propriedades e métodos para qualquer tipo de identificador.	5
Possibilitar a passagem de parâmetros para macros sempre como referência.	8
Implementação de contêineres para diversas classes nativas do Delphi.	1
Implementação de prioridade no mapeamento entre classe e classe contêiner.	3
Implementação da diretiva `#break` para terminar abruptamente o processamento da diretiva de controle de fluxo por repetição (`#foreach`).	4
Implementação da diretiva `#continue` para terminar abruptamente o processamento do laço atual da diretiva de controle de fluxo por repetição (`#foreach`), partindo para o processamento do próximo laço.	4

Obrigado!